MySQL

Comece com o principal banco de dados open source do mercado

VINÍCIUS CARVALHO

Prefácio

Escrevo este livro pensando nas pessoas que estão começando a estudar banco de dados, a trabalhar com ele, e queiram algo bem prático para iniciar; e para os desenvolvedores que necessitam de uma ajuda no dia a dia, que queiram migrar suas aplicações para o MySQL, ou se aperfeiçoar utilizando, na prática, os benefícios deste poderoso Sistema Gerenciador de Banco de Dados (SGBD).

Agradecimentos

Ao lado de um grande homem existe uma grande mulher. E ao meu lado, desde a época em que estava estudando para ingressar na universidade, tenho uma mulher fantástica. Minha musa inspiradora, que sempre apoiou e incentivou minha carreira profissional e minha busca por conhecimento. Muito obrigado.

Gostaria também de agradecer a minha família, principalmente minha mãe. Mesmo não entendendo muito o que eu faço, ela sempre apoiou minhas decisões e me ajudou no que estava ao seu alcance.

A todas as pessoas com que tive a experiência de trabalhar, aprender, ensinar e conviver. Saiba que tentei absorver o melhor de cada uma e todas foram importantes à sua maneira.

Também, a todos os professores que tive em minha vida, pois, sem eles, nada de que já conquistei profissionalmente seria possível. Sempre me mostraram o caminho para que eu conseguisse buscar conhecimento.

Por último, não poderia deixar de agradecer à Casa do Código, que me proporcionou esta maravilhosa experiência de escrever este livro. Muito obrigado!

Sobre o autor

Vinicius Carvalho teve seu primeiro contato com o computador em um curso de MS-DOS com Windows 95 e, desde então, apaixonou-se pela computação. Ao longo da adolescência, procurou aperfeiçoar-se e fazer alguns cursos até chegar a hora de escolher sua formação na faculdade. Essa parte foi fácil! Formou-se em Sistemas de Informações, pós-graduou-se em Engenharia de Software e não parou de aprender coisas novas.

Apaixonado pela busca pelo conhecimento, procura manter-se atualizado nas tendências de desenvolvimento de software, tecnologia e tem como meta aprender algo novo todos os dias.

Na sua carreira profissional, teve oportunidades de trabalhar como analista de suporte, desenvolvedor, gerente de projetos, consultor e como um empreendedor incansável, arriscando-se a ter seu próprio negócio. Atualmente é analista de sistemas sênior do maior grupo de ensino do mundo, a Kroton, além de também fazer algumas consultorias na área de desenvolvimento de software e participar de grupos de discussão sobre empreendedorismo em sua cidade.

Teve chance de palestrar em congresso de software livre como o VOL DAY, evento criado pela comunidade Viva o Linux; publicar artigos em diversos congressos no Brasil; e ministrar aulas de graduação no Centro Universitário Filadélfia (UniFil), faculdade que é referência em cursos de graduação e pós-graduação no Paraná, na qual se formou.

Sua página pessoal é http://www.viniciuscdes.net. Lá você pode conferir seu currículo e o link para seu blog, que aborda diversos temas, como: tecnologia, computação, produtividade, informação, entre outros.

Sumário

Versão: 18.6.27

Capítulo 1

Introdução

"Para mim, o computador é a mais extraordinária ferramenta que já tivemos. É o equivalente à bicicleta para nossa mente."
– Steve Jobs

Quando vamos iniciar o desenvolvimento de um novo projeto, um grande ponto de interrogação surge em nossa cabeça a respeito de qual tecnologia utilizar. Algo que deve ser levado em consideração é o seu orçamento. Escolher ferramentas livres de taxas, de qualidade e que o suportem é de grande importância.

Ao escolher o MySQL como opção de Sistema Gerenciador de Banco de Dados (SGBD), além de uma ferramenta gratuita criada na base da licença de *software* livre, você também está optando por qualidade, robustez e segurança. Estes são adjetivos que um gerenciador deve ter, pois guardar seus dados ou de seus clientes com segurança é o mais importante.

Além das ferramentas, você também deve preocupar-se com o planejamento do projeto que está desenvolvendo. A modelagem e a construção do banco de dados de um sistema é o coração dele. O banco de dados vai impactar o processo inteiro: desde o início da criação do projeto, o desempenho do sistema durante seu desenvolvimento e até sua manutenção e expansão posteriormente. Por isso, volto a frisar a importância de uma boa modelagem e de um bom gerenciador.

É isto que farei ao decorrer deste livro: apresentar a modelagem do projeto, boas práticas, tudo isso de uma forma bem prática, para que, no final, você seja capaz de criar um banco de dados relacional para qualquer aplicação ou sistemas comerciais que deseja desenvolver.

1.1 Sobre o MySQL

Quando você digita 'MySQL' no Google, o primeiro resultado mostra que ele é o banco de dados *open source* mais popular do mundo. Preciso dizer mais alguma coisa? As maiores empresas de tecnologia utilizam e muitas delas contribuem para o projeto. Em vez de escrever algo teórico ou histórico para explicar o que é o MySQL e elencar suas qualidades, eu escolhi criar um mapa mental para listar suas vantagens de forma clara e visual. Você pode acessar o link http://goo.gl/nuRUQ9 para ler um pouco mais sobre essas características e, no demais, deixo a parte histórica como dever de casa para você pesquisar.
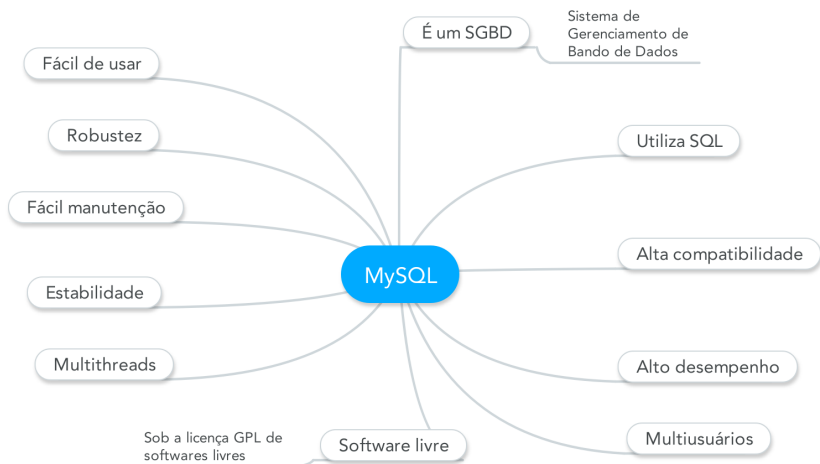
Fig. 1.1: Características do MySQL

1.2 Banco de dados

Apesar de **banco de dados** ser um termo técnico, a maioria das pessoas nos dias de hoje tem contato direto com ele. De fato, grande parte da população atualmente tem acesso a equipamentos, cuja função (principal ou secundária) é o armazenamento de informações. Quem, hoje em dia, não usa um telefone celular?

Desde o seu surgimento, esse tipo de aparelho possui uma agenda, na qual podemos gravar nomes e telefones para, em um segundo momento, acessá-los. Uma lista telefônica impressa também é um exemplo válido disso, pois nela são relatados todos os nomes, endereços e números de telefone das empresas e dos moradores da sua cidade e, eventualmente, dos arredores.

Tudo isso remete ao conceito de banco de dados, ou seja, **um local no qual é possível armazenar informações para consulta ou utilização, quando ne-**

cessário. O próprio banco vai gerenciar a estrutura dos registros e se encarregará de criar espaço para novos registros, alterando seu conteúdo de acordo com as solicitações da aplicação que o está acessando.

Esses bancos de dados que gerenciam os registros de forma automatizada, além de serem apenas um conjuntos de dados, são chamados Sistemas Gerenciadores de Banco de Dados Relacionais (SGBDR), ou *Relational Database Management Systems (RDMS)*.

Há diversas razões para o modelo de banco de dados relacional ser o mais utilizado entre outros modelos existentes. Uma delas é a facilidade da alteração da estrutura das tabelas, como adicionar e excluir colunas e linhas de acordo com as necessidades, sem comprometer sua funcionalidade.

Introdução ao banco de dados relacional

Independentemente do aplicativo que se deseja usar para o armazenamento e manipulação das informações, todos os bancos de dados são constituídos por elementos básicos: **campos**, **colunas**, **linhas** ou **tuplas** e **tabelas**. Campos são os espaços reservados para inserção de um determinado dado; as colunas são os registros de um determinado campo; as tuplas são as linhas de registros de um conjunto de campos; e as tabelas são os conjuntos de linhas, campos e colunas. Para visualizar melhor, se tivéssemos uma tabela de clientes em nosso banco, seria da seguinte maneira:

Fig. 1.2: Composição de uma tabela

Cada banco é um conjunto de tabelas relacionadas. Também são chamados de **relações**, daí o nome **banco de dados relacional**. Cada tabela é uma representação física de uma entidade ou objeto que está em um formato tabular, como vimos anteriormente na figura 1.2.

Como todos os bancos de dados, o relacional também tem sua estrutura baseada em registros relacionados e organizados em tabelas. Essas relações tornam os registros integrados. Esse relacionamento é possível através das chaves: primária (*primary key* PK), estrangeira (*foreign key* FK) e da chave candidata ou alternativa, que vou explicar mais à frente.

Introdução à linguagem SQL

SQL significa *Structured Query Language* e é a linguagem padrão utilizada pelos banco de dados relacionais. Os principais motivos disso resultam de sua simplicidade e facilidade de uso. Mais uma vez não entrarei no mérito histórico; mas algo relevante que você precisa conhecer são suas categorias de comandos. Alguns autores divergem entre exatamente quais são. Eu separei 3. Você pode encontrar ao pesquisar que alguns comandos citados por mim em uma categoria talvez estejam em outra, em um estudo diferente. Elas são:

- **DML Linguagem de Manipulação de Dados**: esses comandos indicam uma ação para o SGBD executar. Utilizados para recuperar, inserir e modificar um registro no banco de dados. Seus comandos são: `INSERT`, `DELETE`, `UPDATE`, `SELECT` e `LOCK`;
- **DDL Linguagem de Definição de Dados**: comandos DDL são responsáveis pela criação, alteração e exclusão dos objetos no banco de dados. São eles: `CREATE TABLE`, `CREATE INDEX`, `ALTER TABLE`, `DROP TABLE`, `DROP VIEW` e `DROP INDEX`;
- **DCL Linguagem de Controle de Dados**: responsável pelo controle de acesso dos usuários, controlando as sessões e transações do SGBD. Alguns de seus comandos são: `COMMIT`, `ROLLBACK`, `GRANT` e `REVOKE`.

> Cada um dos comandos aqui citados será explicado ao longo do livro e aplicado em nosso projeto!

1.3 Começando a utilizar o MySQL

Neste livro, utilizaremos a versão MySQL Commuty Server 5.6. Atualmente, há versões para *download* para 9 plataformas do Linux, Windows e Mac OS. Todos poderão ser feitos no site http://dev.mysql.com/downloads/mysql/. Escolha sua plataforma e o tipo do seu sistema operacional (32 ou 64 bit).

Instalação e configuração no Windows

Para o Windows, há possibilidade de baixar a versão **Windows(x86, 640bit), MySQL Installer MSI** e será este que vou instalar, pois ele fornece um assistente de instalação que facilita bastante, além de downloads extras que poderão ser úteis. Se você estiver começando agora no mundo de MySQL, esta opção será bastante proveitosa. Você precisará de conexão com a internet durante a instalação.

Depois de ter feito o download, execute o arquivo. Logo em seguida, aparecerá uma tela para você aceitar os termos de utilização do MySQL, na qual

você deve clicar na caixa *I accept the license terms* e no botão *Next* para avançar para a próxima tela.

Como fizemos o download desse tipo de instalação, podemos escolher o que queremos instalar na tela a seguir.
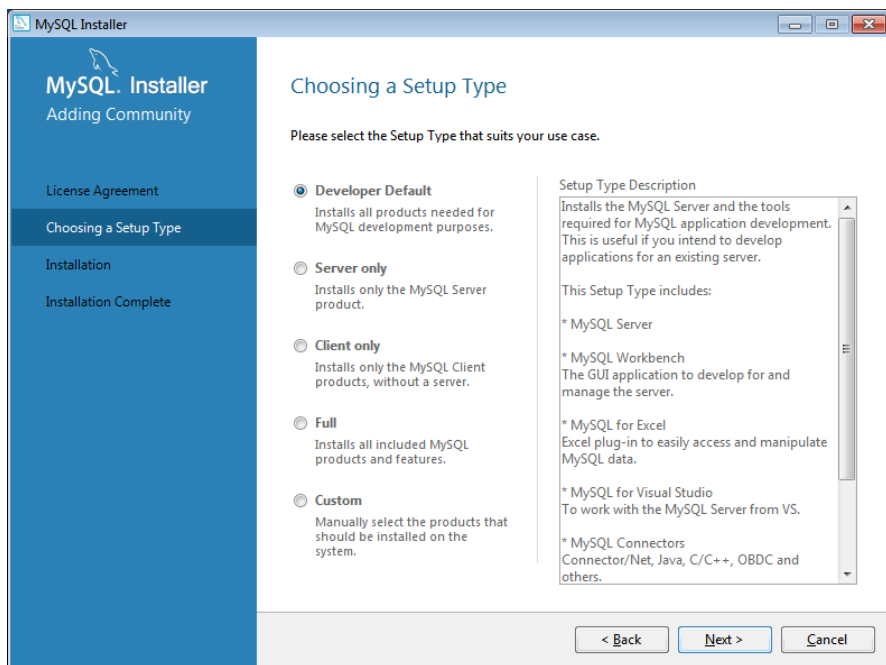
Fig. 1.3: Instalação: versão de instalação

As opções são:

- **Developer Default**: tudo o que precisamos para iniciar a trabalhar com o MySQL. Além do servidor, ao escolher essa opção, podemos também instalar o MySQL Workbench, que é uma *IDE (Integrated Development Environment)* para o banco de dados para trabalhar com SQL, como também possibilitar a criação de entidade e relacionamento, como veremos mais à frente;
- **Server Only**: instala apenas o servidor do MySQL. Prefira esta opção

quando você for colocar o banco de dados em uma rede. Assim, este servidor deverá ser acessado através da versão *Client*;

- **Client Only**: instala a versão que você deve implantar nas máquinas que vão acessar o servidor na rede. Com esta versão, você não conseguirá criar um banco de dados, apenas acessar algum existente;
- **Full**: instala todos os produtos que estiverem disponíveis. Além de instalar o *Server* e *Client*, também instalará as bibliotecas necessárias para conexão de algumas linguagens de programação, como, por exemplo, as bibliotecas para acessar o MySQL utilizando o Java;
- **Custom**: com esta opção, você poderá escolher manualmente quais produtos quer instalar. Se você já estiver familiarizado com essas ferramentas, poderá escolher aquelas que realmente vai utilizar.

Vamos escolher a primeira opção: *Developer Default*, pois nosso intuito é desenvolver e utilizar o banco da mesma máquina, neste primeiro momento.

A próxima tela mostrará quais produtos serão instalados.
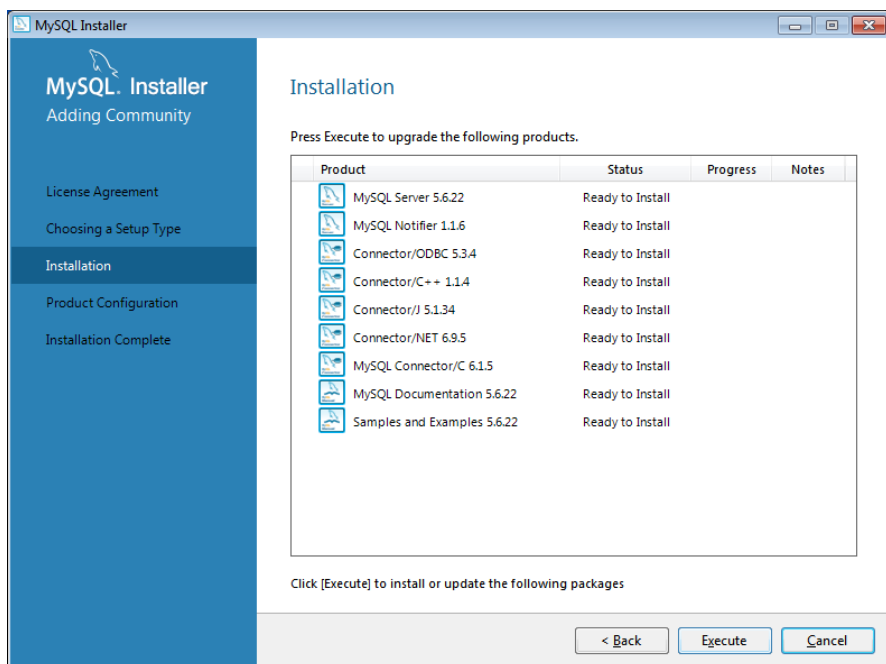
Fig. 1.4: Instalação: produtos a serem instalados

Estou instalando a versão 5.6. Pode ser que, quando você for instalar, outros produtos estejam disponíveis. Agora, é só clicar em *Execute*.

Na tela seguinte, inicia-se a configuração do seu gerenciador. Em *Type and Networking*, você deve dizer em qual máquina você está instalando: em uma máquina de desenvolvimento ou em uma que será 100% dedicada ao gerenciador de banco de dados. Escolheremos a primeira opção, **Development Machine** (máquina de desenvolvimento).

Nesta mesma tela, devemos configurar qual será o tipo de conexão e porta utilizada. Normalmente, estará selecionado, *TCP/IP* e *Port Number* com 3306. Vamos deixar nesta mesma porta, a não ser que você tenha alguma aplicação já a utilizando. Se estiver tudo bem, clicamos em *Next*. A tela estará da seguinte maneira:
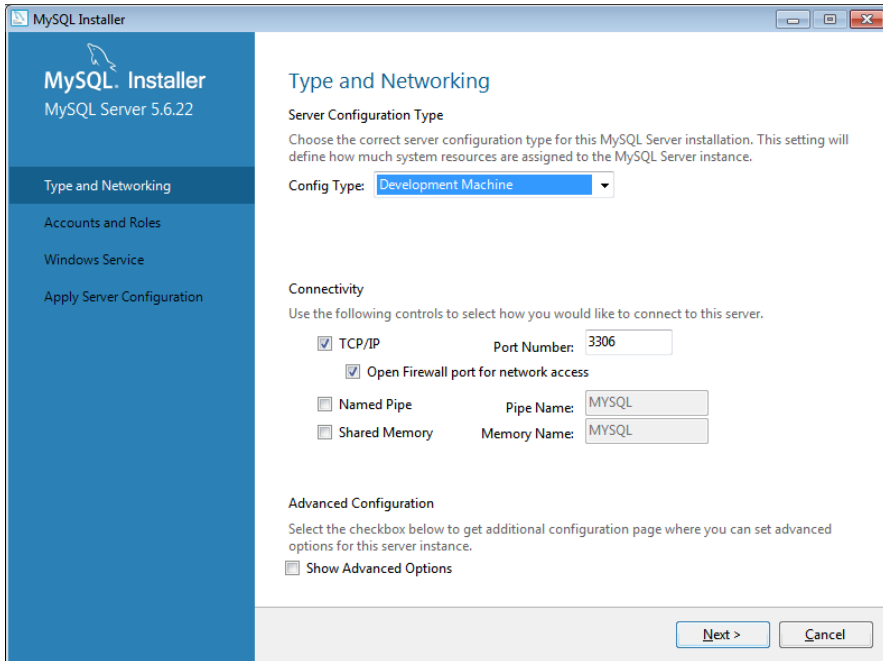
Fig. 1.5: Instalação: tipo de máquina e rede

Na sequência, temos a configuração da senha padrão para o usuário `root`, que é o usuário principal do gerenciador. Nos campos *MySQL Root Password* e *Repeat Password*, vamos colocar como senha **cursomysql**. Você pode escolher a senha que desejar, eu estou padronizando esta que será a mesma para todo o projeto. Você também pode criar outros usuários para acessar seu banco de dados, porém, nesta etapa, ficaremos apenas com o usuário `root`. A criação de usuários e permissões serão apresentadas no capítulo 2. Com isso, a tela estaria da seguinte maneira:
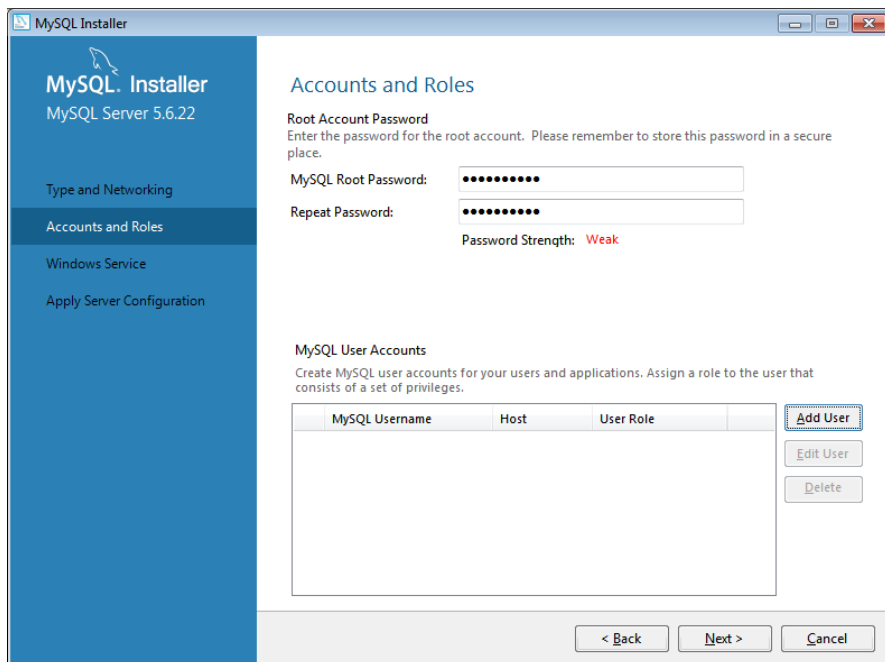
Fig. 1.6: Instalação: usuário root

Agora, na última tela de configuração, deixamos as configurações padrões, clicamos em *Next* e vamos para a próxima, que mostrará os passos a serem executados pelo assistente de instalação. Clicando em *Execute*, as ações necessárias serão feitas e seu gerenciador de banco de dados será instalado.

Instalação e configuração no Linux (Ubuntu)

Para a instalação no Linux, utilizaremos o Ubuntu. Você pode optar por baixar a versão para esse sistema no site do MySQL supracitado ou diretamente no seu gerenciador de pacotes. Esta é a forma mais simples de instalar.

Atualize o gerenciador de pacotes com:

```
$> sudo apt-get update
```

Após atualizado, podemos baixar e instalar o `mysql server`:

```
$> sudo apt-get install mysql-server
```

Ao finalizar a instalação, para você abrir o MySQL e começar a criar suas tabela, digite:

```
$> sudo mysql -u root -p
```

Mais à frente, explicarei como configurar uma senha para o usuário `root`.

Instalação e configuração no Mac-OS

Para a instalação no Mac, escolhi o pacote DMG, mas fique à vontade para escolher outra versão. Eu estou usando o Mac OS X 10.9. Agora, com o download feito, vamos à instalação.

Nessa versão que baixei, devemos montar em forma de disco. Este pacote terá os arquivos que utilizaremos na instalação. Primeiro, vamos instalar o arquivo, `mysql-5.6.22-osx10.9-x86_64.pkg`. Ao clicar nele, poderá surgir uma mensagem de erro, dizendo que você não tem permissão para instalar programas de desenvolvedores desconhecidos, como mostra a figura 1.7.

Fig. 1.7: Permissão para instalar software de desenvolvedores desconhecidos

Para resolver este problema, vá até *System Preferences > Security & Privacy*. Quando abrir uma janela, clique no cadeado que se encontra no canto esquerdo inferior para desbloquear, e marque a opção *Anywhere* da lista que

diz *Allow apps downloaded from*. Em seguida, abrirá um *pop-up* para você confirmar a permissão. Desta vez, clique no botão *Allow From Anywhere*, como mostra a imagem 1.8.
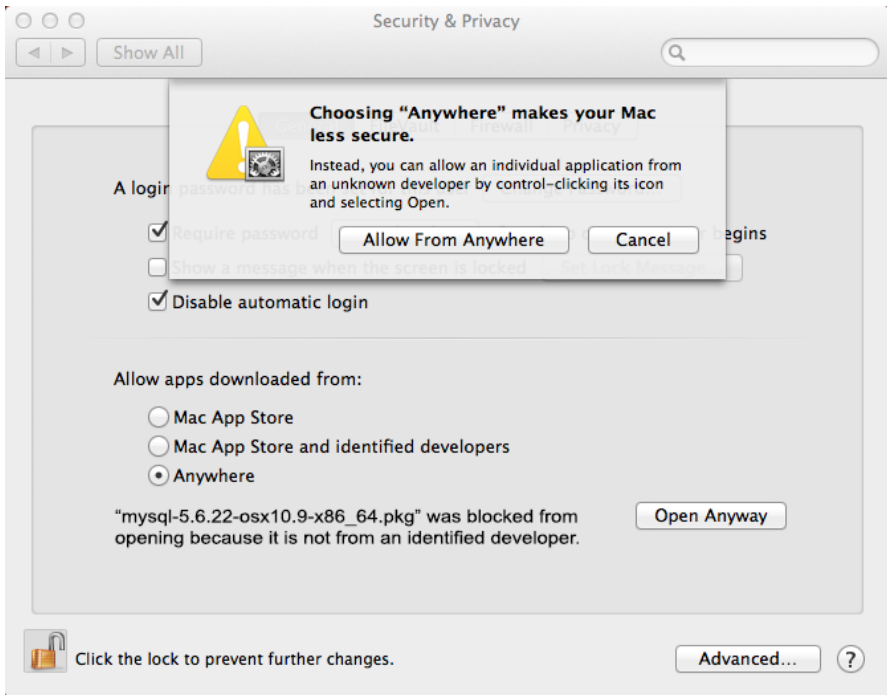
Fig. 1.8: Configuração e segurança

Feito isso uma vez, você conseguirá instalar qualquer software baixado da internet. Feche a janela, volte até o nosso arquivo de instalação e execute-o. A instalação é bem simples, siga as instruções e aceite os termos de licença que aparecerem nas janelas, até receber a mensagem de conclusão.

Junto ao nosso pacote de instalação, há o arquivo `MySQL.prefPane`. Ele vai instalar um painel de configurações para auxiliar o início e/ou a finalização do serviço do MySQL Server e também a configurar sua inicialização automática. Dois cliques no arquivo e a tela, como vemos na figura 1.9, abrirá.
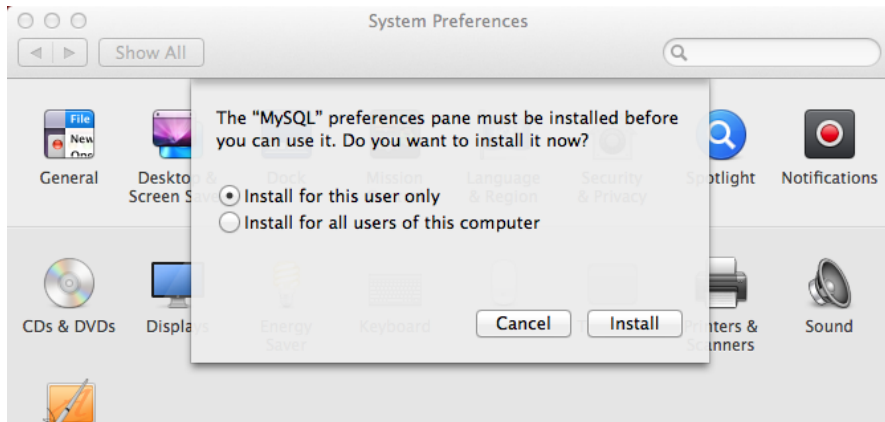
Fig. 1.9: Painel de configurações do MySQL

Clique no botão *Install* para iniciar a instalação. Após a conclusão, vai aparecer um novo icone do MySQL em *System Preferences*. Clique nele e, em seguida, em *Start MySQL Server*. Pronto, o serviço do seu servidor MySQL já esta rodando e está pronto para você começar a trabalhar! Para abri-lo, vá até o terminal e digite o seguinte comando:

```
$ /usr/local/mysql/bin/mysql -u root -h localhost -p
```

Se aparecer a mensagem *Enter Password*, apenas tecle *enter* e a mensagem de boas-vindas surgirá. No capítulo 2, como falei anteriormente na instalação da versão Linux, explicarei como adicionar uma senha para o usuário `root`.

Daqui para frente, tanto no Linux, Windows e MacOS, os comandos serão os mesmos. Durante o projeto, eu utilizarei o Windows, mas não se preocupe. Eu mostrarei como proceder nos outros sistemas operacionais para caso surgir algo diferente.

Concluímos a primeira missão: a instalação! Já poderíamos criar um banco de dados e as tabelas. Porém, antes, precisamos conhecer alguns outros conceitos e especificar um pouco mais o projeto que vamos desenvolver. Até o próximo capítulo.

Capítulo 2

Iniciando o projeto

"Faça as coisas o mais simples que você puder, porém não as mais simples."
– Albert Einstein

2.1 Criando nosso primeiro banco de dados

Fomos contratados por algum cliente para desenvolver um sistema para vendas. No 3 descreverei mais detalhes sobre o projeto. Como usaremos o mesmo exemplo em todo o livro, também utilizaremos um único banco de dados.

Quando instalamos e configuramos o MySQL, nós criamos o usuário `root` e configuramos uma senha. Com eles, nós podemos criar nosso banco de dados. Para acessar o MySQL, utilizaremos o console que se instala junto à instalação que fizemos. No Windows, ele fica na mesma estrutura de pastas do MySQL e se chama `MySQL 5.6 Command Line Client`, na versão

que estou utilizando.

Ao clicar, abrirá uma tela preta, igual à tela do *prompt* do Windows. Ela solicitará sua senha, aquela configurada na instalação, como mostra a figura 2.1.
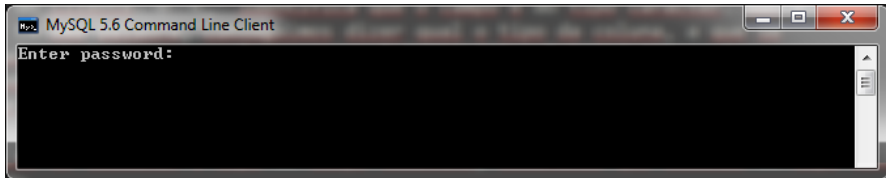
Fig. 2.1: Console do MySQL

Ao realizar o login com sucesso, deverá aparecer a mensagem de boas-vindas do MySQL e o cursor do mouse estacionado sobre a linha `mysql>`, aguardando seus comandos.

```
MySQL 5.6 Command Line Client
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.6.21-log MySQL Community Server (GPL)

Copyright (c) 2000, 2014, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

Fig. 2.2: Login no prompt

2.2 Criando e manipulando usuários

Por padrão, temos o usuário `root` para acessar o MySQL e trabalharmos. Podemos continuar utilizando-o ou podemos criar um novo. Como as boas práticas de desenvolvimento de software aconselham a criação de usuários diferentes do `root` para acessarmos os bancos de dados e para utilização por aplicações, seguiremos esse conselho. Além disso, elas também orientam a criação de, no mínimo, três bancos de dados para um projeto: um de desenvolvimento, um para testes e um outro para produção. Entretanto, em nosso projeto, vamos criar apenas um banco e um novo usuário para acessá-lo.

O nosso novo usuário terá o nome de `usermysql` e sua senha será `cursomysql`. Vamos utilizar o comando `create user`, da seguinte maneira:

```
mysql> create user usermysql@'%' identified by 'cursomysql';
```

Quando utilizamos o `%` em nosso código, estamos dizendo que este usuário poderá acessar o nosso banco a partir de qualquer *host*. Poderíamos ter limitado ao acesso do local apenas, substituindo o `%` por `localhost`. Ele já está criado, porém não tem nenhuma permissão. Como não precisamos limitá-lo, vamos conceder direito total a ele. Faremos isso com o seguinte comando:

```
mysql> grant all privileges on *.* to usermysql@'%'
with grant option;
```

Utilizamos `grant` para conceder o acesso de usuários. Porém, se quiséssemos revogá-lo, faríamos da seguinte maneira:

```
mysql> revoke all on *.* from usermysql;
```

Controle de acesso

Quando trabalhamos com desenvolvimento de software, poderá surgir a necessidade de dar acesso a alguma pessoa ou aplicação, no banco de dados. Para não liberar um acesso completo, você utiliza os direitos de usuário para fazer esta limitação.

Os comandos `grant` e `revoke` permitem os administradores do sistema criar usuários e conceder e revogar direitos aos usuários do MySQL em seis níveis de privilégios:

- **Nível global**: privilégios globais aplicam-se para todos os bancos de dados em um determinado servidor. São concedidos e revogados por meio dos comandos a seguir, que concederão e revogarão apenas privilégios globais, respectivamente:

```
mysql> grant all on *.* to usermysql@localhost;
mysql> revoke all on *.* from usermysql;
```

- **Nível dos bancos de dados**: privilégios de bancos de dados aplicam-se a todas as tabelas em um determinado banco de dados. Os comando para conceder e revogar apenas privilégios de banco de dados serão:

```
mysql> grant all to comercial.* to usermysql@localhost
mysql> revoke all on comercial.*;
```

- **Nível das tabelas**: privilégios de tabelas aplicam-se a todas as colunas em uma determinada tabela. São concedidos ou revogados utilizando os comandos:

```
mysql> grant all on comercial.nome_tabela;
mysql> revoke all on comercial.nome_tabela;
```

- **Nível das colunas**: privilégios de colunas aplicam-se a uma única coluna em uma determinada tabela. Podem ser utilizados para os comandos de seleção, inserção e atualização de determinadas colunas que desejar. São concedidos utilizando os comandos:

```
mysql>grant select(nomecoluna1),
            insert(nomecoluna1),
            update(nomecoluna1)
          on comercial.nome_tabela
          to usermysql@localhost
          identified by senha;
```

- **Nível stored routine**: a rotina de alterar, criar rotina, executar e privilégios de concessão de opção aplica-se a *stored procedures* (procedimentos e funções). Eles podem ser concedidos aos níveis globais e de banco de dados. Também podem ser usados no nível de rotina para rotinas individuais, exceto para criar uma. Se você não sabe o que é uma *store procedure*, não se preocupe. No capítulo 6, você verá várias explicações sobre o assunto. Esses privilégios são concedidos ou revogados utilizando os comandos:

```
## para rotinas
mysql> grant routine on comercial.* to usermysql@localhost;
## para procedures
mysql> grant execute on procedure comercial.nomeprocedure
       to usermysql@localhost;
```

- **Nível proxy user**: o privilégio de *proxy* permite que um usuário seja proxy de outro. O usuário externo de um outro host assume os privilégios de um usuário. Utilizando os comandos:

```
mysql> grant PROXY on usermysql@localhost to
'usuarioexterno'@'hostexterno';
```

Como já temos o usuário que vamos utilizar durante o projeto, devemos conectar ao MySQL usando-o.

Para isso, devemos abrir o prompt do Windows e navegar até a pasta `bin` da instalação do MySQL, que, no meu caso, está na pasta `c:\mysql\bin`.

```
c:\mysql --user=root -psenha
```

Agora, conectado com o novo usuário, podemos criar o nosso primeiro banco de dados!

2.3 Criando nosso banco

Daremos o nome ao nosso projeto de `Comercial`. Geralmente, nomeamos o banco de dados com nome ou função executada pelo sistema. Desta maneira, ele também se chamará `comercial`.

```
mysql> create database comercial;
mysql>
```

> **Atenção!** O Windows não é *case sensitive*, ou seja, ele faz distinção entre maiúscula e minúscula; mas alguns sistemas operacionais baseados em Unix são. Se no futuro você desejar migrar seu banco para outro sistema, isso pode causar sérios problemas. Por isso, desde o início, utilize apenas letras maiúsculas ou minúsculas. Utilizarei apenas as minúsculas durante todo o projeto. Existem maneiras de se contornar esses problemas usando variáveis do MySQL e do próprio sistema operacional Unix. No capítulo 11, voltarei a este assunto e explicarei como proceder.

Para verificar se o banco foi criado com sucesso, utilize o comando `show databases` da seguinte maneira:
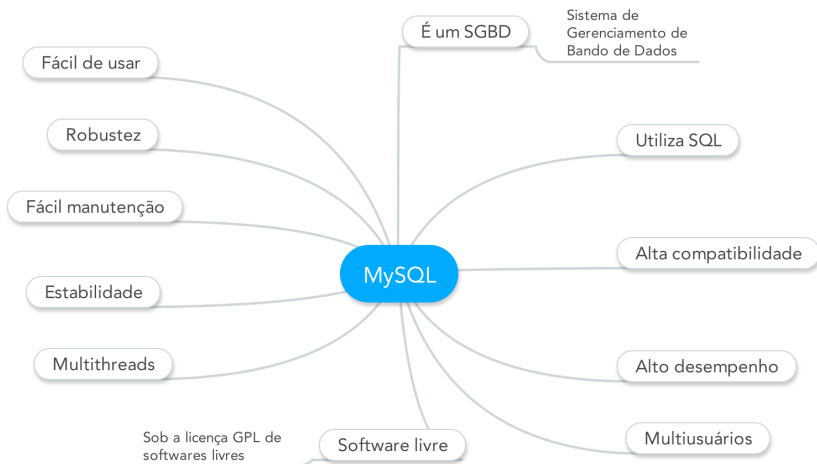
Fig. 2.3: Características do MySQL

Verificando isso, temos que dizer para ao SGBD que queremos usar o banco, com o seguinte comando:

```
mysql> use comercial;
```

Pronto! Agora poderemos criar as tabelas. É hora de planejar o que precisamos criar para o projeto.

2.4 Requisitos para o projeto

No decorrer deste livro, vamos simular que fomos contratados para desenvolver um projeto para um cliente, que solicitou um sistema para vendas de produtos. Nesse sistema, ele gostaria de fazer os seguintes cadastros:

- Clientes
- Fornecedores
- Vendedores
- Produtos
- Vendas

Não há nada melhor para aprender a programar do que começar um projeto prático e de preferência que tenha alguma aplicação real. Dessa forma, você consegue iniciar o aprendizado pelo básico e ir gradualmente adicionando elementos mais complexos, uma vez que os sistemas continuam evoluindo e tornando-se mais complicados.

Nas boas práticas de desenvolvimento e engenharia de software, após o levantamento dos requisitos, a próxima etapa é o desenvolvimento da modelagem do banco de dados, que consiste na sua criação. Isso pode evitar alguns problemas que podem por em risco seu projeto. Algum deles são a falta de campos na tela e, o pior, a inconsistência de dados. Uma vantagem de se fazer a modelagem antes das telas é a agilidade que ganharemos ao desenvolvê-las, pois os campos já foram definidos anteriormente.

2.5 (Minhas) Boas maneiras

Depois de alguns anos programando, você acaba desenvolvendo manias e métodos próprios. E não há nada melhor do que um banco de dados padronizado e organizado. Para isso, adoto um padrão sempre que vou fazer a modelagem de um novo banco de dados, pois fica mais fácil a leitura das consultas posteriormente. Além disso, o aprendizado para novas pessoas que forem trabalhar no mesmo banco que você torna-se muito mais fácil.

Repositórios

Repositórios são softwares que fazem o controle de versão de seus arquivos. Por exemplo, você tem um arquivo de texto, fez uma alteração nele e o salvou. Depois de um mês e de várias outras alterações, você deseja saber o que foi alterado nele desde a primeira vez que você escreveu. Se ele estivesse versionado em algum repositório, você poderia consultar todas suas mudanças. Essa é a função do repositório.

Caso você queria conhecer um pouco mais sobre isso, você pode adquirir o livro *Controlando versões com Git e GitHub* da Casa do Código (http://www.casadocodigo.com.br/products/livro-git-github) sobre Git e Github, um dos repositórios mais utilizados no mundo.

Algo que eu também tenho como padrão de desenvolvimento é fazer o versionamento de todos os arquivos e scritps gerados durante um projeto. Todos os arquivos que eu citar no livro estão disponíveis em meu repositório particular no GitHub. Você pode acessá-lo através do link: https://github.com/viniciuscdes/mysqlbook. Veja a figura . O programa mostra todas as alterações que fiz no arquivo `popula_banco.sql`.
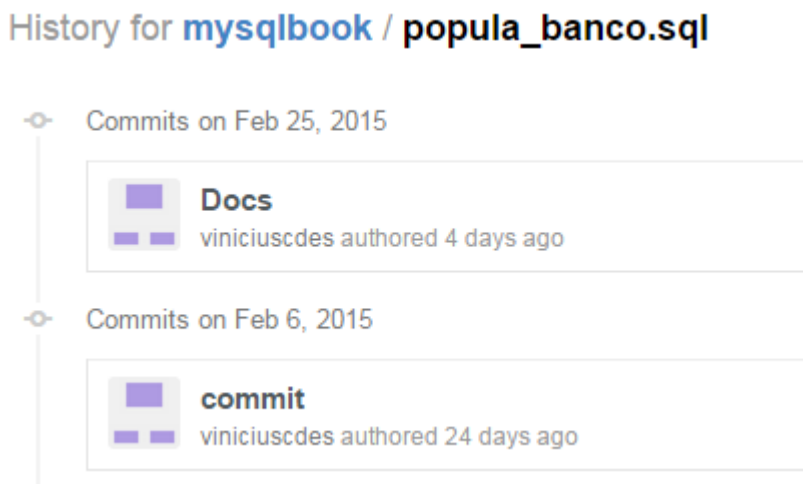
Fig. 2.4: GitHub armazena todas as modificações de seus arquivos

Padronização do nome das tabelas

Para criação dos nomes das tabelas, eu faço uma relação com o sistema e com aquilo a que ela vai se referir. Em nosso sistema `Comercial`, a tabela de clientes ficaria `COMCLIEN`. Se ele se chamasse `Financeiro`, ela seria `FINCLIEN`.

Não fica mais fácil para saber de qual sistema faz parte e a que a tabela se refere? Assim, podemos montar o esqueleto do nosso padrão. Além da uniformização dos nomes, padronizo também o número de letras: apenas 8 caracteres para a criação das tabelas. Volto a repetir que esta é uma regra que eu utilizo. Logo, caso queira, você pode seguir o seu padrão para ambos os casos.

Tabela: COMCLIEN

COM -> Identifica sistema comercial
CLIEN -> Identifica cadastro de clientes

Fig. 2.5: Padrão para criação de nome das tabelas

Pensando sempre no longo prazo, quando seu sistema atingir um grande número de tabelas, essa padronização fará toda a diferença, pois ficará mais fácil saber o que as consultas estão querendo dizer, como também a manutenção no banco de dados.

Padronização do nome dos campos

Além de uniformizar as tabelas, vamos padronizar também seus nomes dos campos. Esta padronização é até mais importante do que a anterior, pois, além de saber a qual tabela o campo pertence, conseguiremos ver qual o seu tipo: se é um campo caractere, um campo numérico etc.

Ao olhar para uma consulta pela primeira vez com n campos e n tabelas, você fica perdido tentando saber de qual tabela é cada coluna e qual é seu tipo. Utilizando nossa tabela `COMCLIEN`, a coluna de nome do cliente seria `C_NOMECLIEN`. Mesmo sem conhecer a padronização, você já consegue identificar que o campo é da tabela de clientes, além de saber também que se trata do nome dele. Vamos adotar o `C_` para identificar que o campo é do tipo caractere.

Não é interessante? Em uma palavra, conseguimos dizer qual o tipo do campo, a que se refere e a qual tabela pertence. Os campos `N_NUMECLIEN` e `D_DATACLIEN` seriam o número de identificação do cliente e a data do seu cadastro, respectivamente. Muito simples, não? O padrão completo de campos será da seguinte maneira, como apresentado na figura 2.6:

Campo: C_NOMECLIEN

C_ -> Identifica seu tipo
NOME -> Identifica o campo
CLIEN -> Identifica a tabela

Fig. 2.6: Padrão para criação de campos

A letra que identifica o tipo do campo é a letra inicial do nome do tipo. No exemplo, o campo era caractere, logo a letra foi `C_`. Para os demais tipos de dados do MySQL, temos:

- **C_**: para campo do tipo caractere;
- **D_**: para campo do tipo data;
- **N_**: para campo do tipo numérico;
- **B_**: para campo do tipo blob.

Não se preocupe em reconhecer os tipo de dados agora. Na sequência, vamos tratar sobre esse assunto e conhecer cada um.

2.6 Tipos de dados

Conhecer todos os tipos de dados existentes do MySQL é muito importante, uma vez que será algo que vai impactar no funcionamento de seu sistema. Com experiência, ficará automático decidir qual tipo de dado utilizar em cada coluna. No começo, você vai pensar um pouco em qual tipo usar em cada campo, mas não se preocupe. Por exemplo, como você escolheria o tipo do campo para salvar o telefone do cliente? Vale lembrar que campos do tipo

numérico não salvam zeros à esquerda. Este é um dos cuidados que você deve ter.

O MySQL, como a maioria dos outros SGBD, possui 3 categorias de tipos de dados: texto, número e data/tempo.

Tipo texto

- **CHAR(tamanho)** : guarda um número fixo de caracteres. Pode conter letras, números e caracteres especiais. O tamanho deve ser declarado entre parênteses. Guarda até 255 caracteres.
- **VARCHAR(tamanho)**: ele possui as características do tipo `CHAR`, com a diferença de que, se você criar com mais de 255 caracteres, ele transforma para o tipo `TEXT`. Ou seja, se for criar algum campo com mais de 255, já crie como `TEXT`.
- **TEXT**: guarda uma *string*: com o tamanho máximo de 65.535 caracteres.
- **BLOB**: é o tipo de dado medido pela quantidade de bytes, em vez de pela quantidade de caracteres, conforme a maioria. Pode salvar por imagens, por exemplo, com o máximo de 65.535 bytes de arquivo.

Dica: em muito lugares você encontrará exemplos que salvam as imagens diretamente no banco de dados. Mas, em vez de salvá-las nele, prefira utilizar um campo `TEXT` para salvar apenas o caminho em que a imagem se encontra e, por meio da programação de sua aplicação, linká-la. Assim, você ganhará em desempenho de banco de dados, pois não vai salvá-la no banco. Também, se você for desenvolver em duas plataformas que usem bancos de dados distintos, isso facilitará quando quiser recuperá-las.

Tipo numérico

- **TINYINT**: guarda números do tipo inteiro. Suporta de -128 até 127 caracteres.
- **SMALLINT**: guarda números do tipo inteiro. Suporta de -32768 até 32767 caracteres.
- **MEDIUMINT**: guarda números do tipo inteiro. Suporta de -8388608 até 8388607 caracteres.
- **INT(tamanho)**: guarda números inteiros. Suporta de -2147483648 até 2147483647 caracteres. O número máximo de caracteres pode ser especificado entre parênteses.
- **BIGINT**: guarda números do tipo inteiro. Suporta de -9223372036854775808 até 9223372036854775807 caracteres.
- **FLOAT(tamanho,decimal)**: guarda números `REAIS`. O número máximo de caracteres pode ser especificado entre parênteses. Deve-se especificar o tamanho inteiro e o tamanho numérico da coluna.
- **DOUBLE(tamanho,decimal)**: guarda números `REAIS`. O número máximo de caracteres pode ser especificado entre parênteses. Deve-se especificar o tamanho inteiro e o tamanho numérico da coluna. Esse tipo armazena uma quantidade maior de número do que os campos do tipo `FLOAT`.

Fique atento ao colocar a quantidade de casas decimais, pois, se incorreto, afetará os cálculos que seu sistema efetuará.

Tipo date/time

Colunas de data e hora são uma grande pedra no sapato de muito desenvolvedores, pois cada SGBD e cada linguagem de programação tratam de maneiras diferentes. Tenha muito cuidado. Procure conhecer o formato que você vai utilizar no sistema. Para saber qual o formato de data e hora que o seu SGBD exige, consulte o 6, no qual apresento uma função para você verificar isso.

- **DATE()**: tipo de campo que vai armazenar datas no: YYYY-MM-DD, onde Y refere-se ao ano, M ao mês e D ao dia;
- **DATETIME()**: a combinação de data e tempo, no formato YYYY-MM-DD HH:MI:SS;
- **TIME()**: armazena horas, minutos e segundos no formato HH:MI:SS.

Tente utilizar os tipos de dados corretamente para cada tipo de informação. Repetirei várias vezes no livro: **É importante pensar no futuro de sua aplicação**. Pense que ela evoluirá e a complexidade vai aumentar. Um fato curioso que aconteceu em 2013 é que o vídeo *PSY - GANGNAM STYLE* no YouTube estourou a capacidade do campo que apresentava o número de visualizações, obrigando o Google a alterar o tipo desse campo. Imagine o risco para um sistema não prever a capacidade de informações que vai armazenar. Repare que estamos falando do Google.

2.7 Modelando o projeto

Para modelar o banco de dados, existem no mercado vários mecanismos. Eu, particularmente, gosto de utilizar o Workbench. Ele é uma ferramenta visual unificada para arquitetos de banco de dados, desenvolvedores e DBAs.

MySQL Workbench fornece modelagem de dados, desenvolvimento de SQL e ferramentas de administração abrangentes para a configuração do servidor, administração de usuários, *backup* e muito mais. Ele está disponível para Windows, Linux e Mac OS X. Quando fizemos a nossa instalação, se você não desmarcou a opção para instalar o Workbench, ele já deve estar instalado. Se não estiver, você pode baixá-lo pelo link: http://www.mysql.com/products/workbench/.

É comum, ao iniciar-se o aprendizado sobre programação, achar que basta saber uma linguagem de programação e já sair programando sem nenhum planejamento. É assim que muitos projetos falham ou causam problemas, pois não foi feito um estudo do que deveria ser desenvolvido. Isso pro-

vavelmente se deve às pressões por sistemas em prazos cada vez mais curtos e com menores custos de produção.

Porém, por outro lado, isso acaba por prejudicar, e muito, o entendimento correto do problema e, consequentemente, a construção de um sistema que atenda às reais expectativas do usuário. Esta situação muitas vezes origina sistemas de baixa qualidade com elevada necessidade de modificação e de difícil manutenção. Por isso, procure entender e planejar muito bem o que você deverá desenvolver.

Como estamos trabalhando em um projeto no qual vamos criar apenas a parte do banco de dados, devemos pensar na modelagem dos dados. Ela é composta de três etapas.

- **Fase conceitual**: na qual temos um cenário da vida real, e, baseado nele, faremos o levantamento de requisitos que o projeto deve atender. Nesta etapa, devemos explorar todas as necessidades do problema que vamos resolver e, com essas informações, conseguiremos criar um modelo conceitual, que será independente da tecnologia que utilizaremos. Registraremos que dados podem aparecer no banco, mas não como estes dados estão armazenados. Por exemplo: cadastro de clientes (dados necessários: nome fantasia, razão social, endereço, CNPJ, cidade, estado, telefone etc.).

- **Fase lógica**: ao contrário dos modelos conceituais, os lógicos são os modelos em que os objetos, suas características e seus relacionamentos têm suas representações de acordo com as regras de implementação e limitantes impostos por alguma tecnologia. Ele é utilizado já na fase de projeto mais independente de dispositivo físico, implementando conceitos de construção de um banco de dados. Por exemplo: a figura 2.7;

- **Fase física**: elaborada a partir do modelo lógico, leva em consideração limites impostos por dispositivo físico e por requisitos não funcionais dos programas que acessam os dados. Um SGBD diferente poderá definir um modo diferente de implementação física das características e dos recursos necessários para o armazenamento e a manipulação das estruturas de dados. Para exemplificar, apresento-o na figura 2.8 que é o diagrama do nosso projeto.
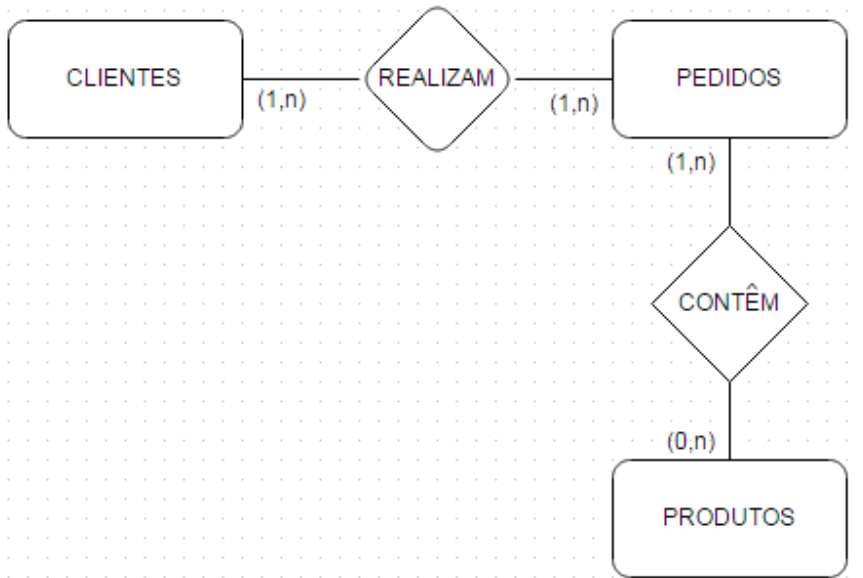
Fig. 2.7: Exemplo de modelo lógico

Não aprofundarei muito nos assuntos sobre modelagem de dados, uma vez que é conteúdo para vários livros. Se você ainda não aprendeu muito sobre o assunto, aconselho a pesquisar, pois isso auxiliará na hora de criar seu projeto.

Diagrama de Entidade e Relacionamento

O Diagrama de Entidade e Relacionamento (DER) do nosso projeto nada mais é que uma representação gráfica (modelo físico) das tabelas do projeto que vamos desenvolver. É muito importante que você desenvolva o DER, pois ficará mais fácil você comunicar de forma visual as alterações no banco de dados para os outros envolvidos nele. O nosso DER inicial está representado na figura 2.8.
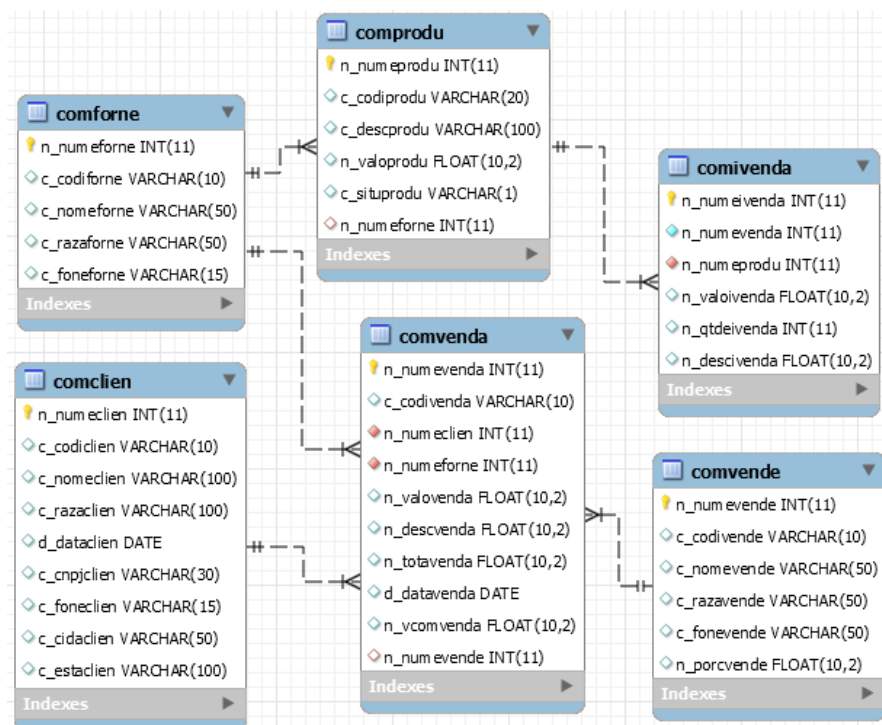
Fig. 2.8: Diagrama de Entidade e Relacionamento

Conforme o projeto cresce, o diagrama deve manter-se atualizado. Assim, você terá o controle visão do projeto.

Chega de teoria, por enquanto. Vamos à prática! Agora que sabemos quais tabelas e campos devemos criar, podemos escrever as instruções para aplicar no banco de dados.

Capítulo 3

Mão na massa: criando nossos códigos

"Sempre escolha uma pessoa preguiçosa para realizar uma tarefa difícil. Ela sempre irá achar a maneira mais simples de se fazer."
– Bill Gates

A fase da criação do banco é uma das mais importantes do processo de desenvolvimento de software. A integridade dos dados dependerá dela. Volto a frisar: gaste um bom tempo nessa etapa!

Como já criamos o banco, agora poderemos criar as tabelas, alterá-las, inserir e manipular registros. Trabalharemos durante todo o livro com as mesmas tabelas.

3.1 Criando as tabelas do projeto

Agora, baseado nas tabelas que nós criamos na modelagem, nós devemos criar os *scripts* (instruções na linguagem SQL), para que elas sejam criadas no banco. A instrução `create table()` é o script que utilizaremos. Entre parênteses, você deve colocar os campos que você quer na tabela, definindo qual o tipo de cada um.

Vamos utilizar os padrões sugeridos anteriormente. A tabela de clientes em nosso padrão fica `comclien`, e, utilizando a instrução de criação, temos o nosso primeiro código para a montagem de uma:

```
mysql>    create table comclien(
                n_numeclien int not null auto_increment,
                c_codiclien varchar(10),
                c_nomeclien varchar(100),
                c_razaclien varchar(100),
                d_dataclien date,
                c_cnpjclien varchar(20),
                c_foneclien varchar(20),
          primary key (n_numeclien));
```

Você pode verificar se realmente a tabela foi criada corretamente utilizando o `desc` (*describe*). Para isso, no terminal, digite:

```
mysql> desc comclien;
```

Algo parecido com a figura 3.1 deve ser mostrado no terminal do MySQL.

```
mysql> desc comclien;
+-------------+--------------+------+-----+---------+----------------+
| Field       | Type         | Null | Key | Default | Extra          |
+-------------+--------------+------+-----+---------+----------------+
| n_numeclien | int(11)      | NO   | PRI | NULL    | auto_increment |
| c_codiclien | varchar(10)  | YES  |     | NULL    |                |
| c_nomeclien | varchar(50)  | YES  |     | NULL    |                |
| c_razaclien | varchar(50)  | YES  |     | NULL    |                |
| d_dataclien | date         | YES  |     | NULL    |                |
| c_cnpjclien | varchar(15)  | YES  |     | NULL    |                |
| c_foneclien | varchar(15)  | YES  |     | NULL    |                |
+-------------+--------------+------+-----+---------+----------------+
7 rows in set (0.02 sec)
```

Fig. 3.1: Descrevendo o conteúdo da tabela de clientes

Observe que, após todos os campos, na última linha, temos: `primary key(n_numeclien)`. Nela estamos informando para o banco de dados o

campo `n_numeclien`, que é a chave primária da tabela e seu registro será único.

Introdução à chave primária

A chave primária é o que torna a linha ou o registro de uma tabela únicos. Geralmente, é utilizada uma sequência automática para a geração dessa chave para que ela não venha a se repetir. Em nosso caso, o `n_numeclien` será único, isto é, nenhum par de linhas possuirá o mesmo valor na mesma coluna. Será uma sequência de preferência numérica que identificará um registro.

> **Dica:** procure usar um campo para chave primária que não seja mostrado na tela de seu sistema. Crie um campo específico para ela e um outro para você poder manipular e mostrar na tela, como o código do cliente, por exemplo. Isso ajudará na flexibilidade do seu sistema, na manutenibilidade e na *performance*.

Auto_increment

A cláusula `auto_increment` é utilizada para incrementar automaticamente o valor da chave primária da tabela. Você pode retornar o próximo valor do campo de outras maneiras, porém com o incremento automático fica mais simples e mais seguro. Por padrão, o `auto_increment` inicia-se do 1. Porém, se houver a necessidade de iniciar por outro valor você pode alterá-lo, fazendo:

```
mysql>     ALTER TABLE comclien AUTO_INCREMENT=100;
```

Da mesma forma que criamos a tabela para **clientes**, faremos para as outras tabelas do nosso projeto.

```
mysql>    create table comforne(
            n_numeforne   int not null auto_increment,
            c_codiforne   varchar(10),
            c_nomeforne   varchar(100),
            c_razaforne   varchar(100),
```

```
            c_foneforne   varchar(20),
            primary key(n_numeforne));

mysql>    create table comvende(
            n_numevende   int not null auto_increment,
            c_codivende   varchar(10),
            c_nomevende   varchar(100),
            c_razavende   varchar(100),
            c_fonevende   varchar(20),
            n_porcvende   float(10,2),
            primary key(n_numeforne));

mysql>    create table comprodu(
            n_numeprodu   int not null auto_increment,
            c_codiprodu   varchar(20),
            c_descprodu   varchar(100),
            n_valoprodu   float(10,2),
            c_situprodu   varchar(1),
            n_numeforne   int,
            primary key(n_numeprodu));

mysql>    create table comvenda(
            n_numevenda int not null auto_increment,
            c_codivenda varchar(10),
            n_numeclien int not null,
            n_numeforne int not null,
            n_numevende int not null,
            n_valovenda float(10,2),
            n_descvenda float(10,2),
            n_totavenda float(10,2),
            d_datavenda date,
            primary key(n_numevenda));

mysql>    create table comvendas(
            n_numevenda int not null auto_increment,
            c_codivenda varchar(10),
            n_numeclien int not null,
            n_numeforne int not null,
            n_numevende int not null,
```

```
            n_valovenda float(10,2),
            n_descvenda float(10,2),
            n_totavenda float(10,2),
            d_datavenda date,
            primary key(n_numevenda));

mysql>    create table comivenda(
            n_numeivenda int not null auto_increment,
            n_numevenda  int not null,
            n_numeprodu  int not null,
            n_valoivenda float(10,2),
            n_qtdeivenda int,
            n_descivenda float(10,2),
            primary key(n_numeivenda));
```

Pronto! Criamos as tabelas do nosso projeto. Observe que `comvendas`, `comivenda` e `comprodu` contêm campos de outras tabelas. É o que chamamos de *foreign key* ou chave estrangeira.

Introdução à chave estrangeira

A chave estrangeira define um relacionamento entre tabelas, comumente chamado de integridade referencial. Esta regra baseia-se no fato de que uma chave estrangeira em uma tabela é a chave primária em outra. Na imagem 1.2 que mostrei no início do livro como exemplo, uma tabela tem o campo `id_estado`, que é uma chave estrangeira. Isto é, ele pode se repetir na tabela de `clientes`. No entanto, deve ser único na tabela de `estados`, pois assim terá uma referência exclusiva. Exemplificando:
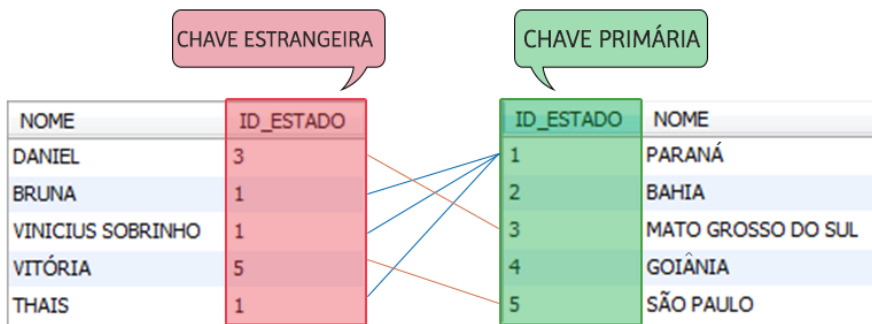
Fig. 3.2: Relacionamento entre duas tabelas

A imagem mostra o relacionamento referencial entre a tabela `clientes` e a `estados`, no qual o campo `id_estado` referencia o campo `id_estado` na tabela `estados`. Assim, podemos identificar de qual estado é cada cliente.

3.2 Cuidando da integridade do banco de dados

Quando criamos a tabela `comvenda`, nós incluímos colunas de outras tabelas, como `n_numeclien`, `n_numeforne` e `n_numeprodu`. Essas colunas estão referenciando um registro em sua tabela de origem. Porém, como apenas criamos o campo, mas nada que informe o banco sobre essa referência, devemos fazer isso, passando uma instrução ao nosso SGBD por meio das `constraints`, como mostram os códigos na sequência.

```
mysql> alter table comvenda add constraint fk_comprodu_comforne
            foreign key(n_numeforne)
                references comforne(n_numeforne)
                    on delete no action
                    on update no action;

mysql> alter table comvenda add constraint fk_comprodu_comvende
            foreign key(n_numevende)
                references comvende(n_numevende)
                    on delete no action
```

```
                    on update no action;

mysql> alter table comvenda add constraint fk_comvenda_comclien
            foreign key(n_numeclien)
                references comclien(n_numeclien)
                    on delete no action
                    on update no action;

mysql> alter table comivenda add constraint fk_comivenda_comprodu
            foreign key(n_numeprodu)
                references comprodu (n_numeprodu)
                    on delete no action
                    on update no action;

mysql> alter table comivenda add constraint fk_comivenda_comvenda
            foreign key(n_numevenda)
                references comvenda (n_numevenda)
                    on delete no action
                    on update no action;
```

Com a criação das `constraints` de chave estrangeira, demos mais segurança à integridade de nossos dados. Agora, se você tentar deletar algum registro da tabela de clientes que possui um registro referenciado na tabela de vendas, o banco de dados barrará a deleção, impedindo que a integridade se perca. Quando declaramos a chave primária em nossas tabelas, o SGBD criará as `constraints` automaticamente.

Se tivéssemos criado uma `constraint` errada, poderíamos deletá-la utilizando a instrução irreversível:

```
mysql>  alter table comivenda drop foreign key
        fk_comivenda_comprodu;
```

3.3 Alterando as tabelas

Com o crescimento de seu sistema, há a necessidade de criação de novas tabelas. Se você reparar em nossa tabela de clientes, não criamos campos para **cidade** ou para **estados**. Para não precisar excluí-la e criá-la novamente, fazemos uma alteração nela com o comando `alter table`.

Acrescentaremos um campo para informar a cidade no cadastro de clientes.

```
mysql> alter table comclien add column c_cidaclien varchar(50);
```

E um campo para informar o estado.

```
mysql>    alter table comclien add column c_estclien varchar(50);
```

Ops! Um erro. Criamos o campo para **estado** fora do padrão. Para criarmos na norma correta, vamos deletar o que geramos errado. Utilizando novamente o comando `alter table`, só que agora com o `drop column`:

```
mysql>    alter table comclien drop column c_estclien;
```

Agora aprendemos a utilizar a instrução `drop column`. Sempre que existir alguma coluna incorreta, podemos deletar. Só não podemos excluir as colunas que são `foreign key` em outra tabela e possuir registros. O SGBD não permitirá isso, para não corromper a integridade dos dados, uma vez que perderá o ponto a que outras tabelas estão referindo-se.

Vamos criar o campo novamente, agora corretamente.

```
mysql> alter table comclien add column c_estaclien varchar(50);
```

Fácil, não? E podemos utilizar o `alter table` para alterar o tipo do campo. Se quiséssemos mudar o tipo do campo `c_estaclien` para o tipo numérico, usaríamos o `alter table` agora com o `modify`. Vamos exemplificar:

```
mysql> alter table comclien modify column c_estaclien int;
```

Porém, lembre de nosso padrão para os campos numéricos e de caracteres. No caso do campo de `c_estaclien`, poderíamos alterar seu tamanho e manter seu tipo como `varchar`, utilizando também o `modify`.

```
mysql>    alter table comclien modify column c_estaclien
       varchar(100);
```

3.4 Excluindo (dropando) as tabelas

Quando criamos nossas tabelas, nós fizemos uma a mais por engano. Foi a tabela `comvendas`, sendo a `comvenda` a correta. Para deletarmos a indesejada, utilizaremos o `drop table`.

```
mysql>   drop table comvendas;
```

Você reparou que usamos o `drop` para excluir qualquer objeto no banco dados? Agora, se você desejar excluir os registros sem excluir a tabela, deverá utilizar uma outra instrução SQL, a qual mostrarei no capítulo 4, entre outros comandos.

Com isso já podemos começar a trabalhar com os dados, pois aprendemos a criar (`reate`), alterar (`alter`), deletar (`rop`) e modificar (`modify`) os objetos no banco de dados.

Este capítulo foi apenas o primeiro da parte prática. Muitos outros virão. Não se preocupe em decorar as instruções SQL. Comece copiando para criar novas tabelas, campos etc. Pratique bastante, que passará a ser algo natural com o tempo.

Capítulo 4

Manipulando registros

"Mova-se rapidamente e quebre as coisas. Ao menos que você não esteja quebrando coisas, você não está se movendo rápido o suficiente."
– Mark Zuckerberg

4.1 Inserindo registros

Aprendemos a modelar, criar, alterar e excluir tabelas. Precisamos agora de registros em nosso banco de dados, pois seu intuito são suas manipulações. Porém, para isso, precisamos aprender como inseri-los através do SGBD, e não por meio de uma aplicação, uma vez que, independente da linguagem de programação em que você estiver trabalhando para desenvolver seu sistema, você poderá inserir registros diretamente no banco de dados através de comando SQL.

Você utilizará essa prática constantemente em sua vida de desenvolvedor, seja para carregar uma tabela com dados novos, em uma migração, para testar algum processo que necessita de informações ou para corrigir algum problema. Com certeza, você vai se deparar com uma situação que demande a necessidade de fazer uma inserção manual.

Vamos fazer o primeiro *insert* na tabela `COMCLIEN` com o comando `insert into COMCLIEN`. Entre parênteses, informaremos em quais colunas queremos inserir os registros. Depois, devemos informar qual o valor para cada coluna, da seguinte maneira:

```
mysql>insert into comclien(n_numeclien,
                           c_codiclien,
                             c_nomeclien,
                           c_razaclien,
                           d_dataclien,
                           c_cnpjclien,
                           c_foneclien,
                           c_cidaclien,
                           c_estaclien)
                   values (1,
                           '0001',
                           'AARONSON',
                           'AARONSON FURNITURE LTDA',
                           '2015-02-17',
                           '17.807.928/0001-85',
                           '(21) 8167-6584',
                           'QUEIMADOS',
                           'RJ');
```

> Quando você executa um comando, tudo está correto e a operação é concluída, uma mensagem do tipo (*Query Ok...*) é mostrada logo após. Se ocorrer um erro com o seu código, será exibida uma mensagem (*ERROR...*). Leia com atenção as mensagem de erros, pois são bem explicativas e ficará fácil para você corrigi-lo.

Muito simples! Se você quiser inserir em todos os campos da tabela, não é necessário descrever quais serão populados. Apenas não se esqueça de con-

ferir se os valores estão na sequência correta, como a seguir, onde omitimos estes campos. O SGBD subentende que todos os campos serão populados.

```
mysql>     insert into comclien
                  values (1,
                          '0001',
                          'AARONSON',
                          'AARONSON FURNITURE LTDA',
                          '2015-02-17',
                          '17.807.928/0001-85,
                          '(21) 8167-6584',
                          'QUEIMADOS',
                          'RJ');
```

> **Lembrete**: você se lembra das `constraints` que criamos no capítulo 3? Quando formos inserir, por exemplo, um cliente na tabela `comvenda`, ele deve estar na tabela `comclien`. Afinal, se ele não existir na tabela, o SGBD retornará um erro e não deixará você inserir, porque quando criamos a `constraint` na tabela de vendas, queremos dizer que deve haver um relacionamento de dados entre ambas. Caso não haja, não conseguiremos consultar os dados dos clientes que estão na tabela `comclien`.

4.2 Alterando registros

Da mesma maneira que conseguimos incluir registros no banco de dados, podemos alterá-los. Uma vez que temos um sistema em produção com pessoas utilizando-o, não podemos excluir os registros para inseri-los corretamente. Por isso, devemos alterá-lo usando o comando `update`.

Você fez a inserção no registro de clientes e errou o nome fantasia. No exemplo que eu descrevi anteriormente, coloquei um incorretamente. Agora, quero corrigi-lo.

```
mysql> update comclien set c_nomeclien = 'AARONSON FURNITURE'
        where n_numeclien = 1;
```

```
mysql> commit;
```

Podemos atualizar mais de um campo de uma vez só, separando com `,`, fazendo:

```
mysql> update comclien set c_nomeclien = 'AARONSON FURNITURE'
                         , c_cidaclien = 'LONDRINA'
                         , c_estaclien = 'PR'
        where n_numeclien = 1;

mysql> commit;
```

Perceba que, além do `update`, utilizei o `set` para informar qual campo que eu quero alterar, o `where` para indicar a condição para fazer a alteração e, em seguida, o `commit` para dizer para o SGBD que ele pode realmente salvar a alteração do registro. Se, por engano, fizermos o *update* incorreto, antes do `commit`, podemos reverter a situação usando a instrução SQL `rollback`, da seguinte maneira:

```
mysql>    update comclien set c_nomeclien = 'AARONSON'
         where n_numeclien = 1;

mysql> rollback;
```

Com isso, o nosso SGBD vai reverter a última instrução. Porém, se tiver a intenção de utilizar o `rollback`, faça-o antes de aplicar o `commit`, pois se você aplicar o `update` ou qualquer outro comando que necessite do `commit`, não será possível reverter.

> **Atenção!** Ao utilizar o `update` para alterar um ou mais registros, não se esqueça de usar o `where` para informar quais registros você deseja mudar. Sem ele, o comando é aplicado a todos registros da tabela.

4.3 Excluindo registros

Incluímos e alteramos registros. Porém, e se quisermos deletar algum? Para isso, devemos utilizar uma outra instrução SQL: o `delete`. Diferente do

drop, ele deleta os registros das colunas do banco de dados. O drop é usado para excluir objetos do banco, como tabelas, colunas, *views*, *procedures* etc.); enquanto, o delete deletará os registros das tabelas, podendo excluir apenas uma linha ou todos os registros, como você desejar.

Desta maneira, vamos apagar o primeiro registro da tabela comclien.

```
mysql> delete from comclien
        where n_numeclien = 1;
mysql> commit;
```

Agora, vamos deletar todos os registros da tabela de clientes.

```
mysql> delete from comcilen;
mysql> commit;
```

Observe que, ao empregar o delete, você também deve usar o commit logo após a instrução. Da mesma maneira, podemos também utilizar o rollback para não efetivar uma deleção de dados incorretos.

Além do delete, podemos fazer a deleção de dados usando uma instrução SQL chamada de truncate. Este é um comando que não necessita de commit e não é possível a utilização de cláusulas where. Logo, só o use se você tem certeza do que estiver querendo excluir, uma vez que ele é irreversível. Nem o rollback pode reverter a operação. Isso ocorre porque, quando você utiliza o delete, o SGBD salva os seus dados em uma tabela temporária e, quando aplicamos o rollback, ele a consulta e restaura os dados. Já o truncate não a utiliza, o SGBD faz a deleção direta. Para usar esse comando, faça do seguinte modo:

```
mysql> truncate table comclien;
```

Lembre-se: nunca se esqueça de criar as constraints de chave estrangeira das tabelas, pois ao tentar excluir um registro, se houver uma constraint nela e ele estiver sendo utilizado em outra tabela, o SGBD não deixará você excluí-lo com intuito de manter a integridade dos dados.

No repositório que citei no início do livro, existe o arquivo `popula_banco.sql`, que possui os scripts de inserção (`insert`) e de alteração (`update`) para você aplicar em seu banco de dados e acompanhar os exemplos no decorrer da leitura.

Inserimos, alteramos e deletamos. Caso você tenha aplicado o arquivo que indiquei ou inserido seus próprios registros, também possuímos vários deles em nosso banco. Agora podemos começar a fazer suas manipulações e seleções.

Capítulo 5

Temos registros: vamos consultar?

"Inspecionar para prevenir defeitos é bom; Inspecionar para encontrar defeitos é desperdício."
– Shigeo Shingo

O objetivo de armazenar registros em um banco de dados é a possibilidade de recuperar e utilizá-los em relatórios para análises mais profundas, processamento dessas informações etc. Essa recuperação é feita através de consultas.

5.1 Estrutura básica das consultas

O comando SQL utilizado para fazer consultas é o `select`. Nada mais óbvio, já que vamos fazemos consultas, ou seja, selecionar dados. Junto ao `select`, devemos dizer ao SGBD de onde você quer selecioná-los; no caso, de qual tabela queremos os registros. Por isso usamos o `from`. Com isso, temos a sintaxe básica para fazer a primeira consulta. Vamos selecionar todos os registros da tabela de cliente.

Quando não queremos selecionar um ou vários campos específicos, utilizamos o asterisco (*). Ficaria da seguinte maneira:

```
mysql> select * from comclien;

+-------------+-------------+-----------------------
| n_numeclien | c_codiclien | c_nomeclien
+-------------+-------------+-----------------------
|           1 | 0001        | AARONSON FURNITURE
|           2 | 0002        | LITTLER
|           3 | 0003        | KELSEY  NEIGHBOURHOOD
|           4 | 0004        | GREAT AMERICAN MUSIC
|           5 | 0005        | LIFE PLAN COUNSELLING
|           6 | 0006        | PRACTI-PLAN
|           7 | 0007        | SPORTSWEST
|           8 | 0008        | HUGHES MARKETS
|           9 | 0009        | AUTO WORKS
|          10 | 00010       | DAHLKEMPER
+-------------+-------------+-----------------------

+-------------------------+-------------+--------------------
| c_razaclien             | d_dataclien | c_cnpjclien
+-------------------------+-------------+--------------------
| AARONSON FURNITURE LTD  | 2015-02-17  | 17.807.928/0001-85
| LITTLER  LTDA           | 2015-02-17  | 55.643.605/0001-92
| KELSEY  NEIGHBOURHOOD   | 2015-02-17  | 05.202.361/0001-34
| GREAT AMERICAN MUSIC    | 2015-02-17  | 11.880.735/0001-73
| LIFE PLAN COUNSELLING   | 2015-02-17  | 75.185.467/0001-52
| PRACTI-PLAN LTDA        | 2015-02-17  | 32.518.106/0001-78
| SPORTSWEST LTDA         | 2015-02-17  | 83.175.645/0001-92
| HUGHES MARKETS LTDA     | 2015-02-17  | 04.728.160/0001-02
```

```
| AUTO WORKS LTDA         | 2015-02-17  | 08.271.985/0001-00
| DAHLKEMPER  LTDA        | 2015-02-17  | 49.815.047/0001-00
+-------------------------+-------------+--------------------
```

```
+----------------+-------------------------+-------------+
| c_foneclien    | c_cidaclien             | c_estaclien |
+----------------+-------------------------+-------------+
| (21) 8167-6584 | QUEIMADOS               | RJ          |
| (27) 7990-9502 | SERRA                   | ES          |
| (11) 4206-9703 | BRAGANÇA PAULISTA       | SP          |
| (75) 7815-7801 | SANTO ANTÔNIO DE JESUS  | BA          |
| (17) 4038-9355 | BEBEDOURO               | SP          |
| (28) 2267-6159 | CACHOEIRO DE ITAPEMIRI  | ES          |
| (61) 4094-7184 | TAGUATINGA              | DF          |
| (21) 7984-9809 | RIO DE JANEIRO          | RJ          |
| (21) 8548-5555 | RIO DE JANEIRO          | RJ          |
| (11) 4519-7670 | SÃO PAULO               | SP          |
+----------------+-------------------------+-------------+

10 rows in set (0.01 sec)
```

Se quiséssemos selecionar apenas o código e a razão social do cliente, no lugar do *, colocaríamos os campos `n_numeclien`, `c_codiclien` e `c_razaclien`.

```
mysql> select n_numeclien, c_codivenda, c_razaclien
        from comclien;

+------------+-------------+-----------------------+
|n_numeclien | c_codiclien | c_nomeclien           |
+------------+-------------+-----------------------+
|          1 | 0001        | AARONSON FURNITURE    |
|          2 | 0002        | LITTLER               |
|          3 | 0003        | KELSEY  NEIGHBOURHOOD |
|          4 | 0004        | GREAT AMERICAN MUSIC  |
```

```
|          5 | 0005        | LIFE PLAN COUNSELLING |
|          6 | 0006        | PRACTI-PLAN           |
|          7 | 0007        | SPORTSWEST            |
|          8 | 0008        | HUGHES MARKETS        |
|          9 | 0009        | AUTO WORKS            |
|         10 | 00010       | DAHLKEMPER            |
+------------+-------------+-----------------------+
10 rows in set (0.00 sec)
```

Ainda podem surgir situações que necessitem selecionar apenas um registro. Neste caso, utilizamos o `where`, da mesma maneira que o usamos no capítulo anterior.

Vamos selecionar o cliente com uma cláusula que deve ter `c_codiclien = '00001'`. Note que coloquei o código dele entre aspas simples. Devemos fazer dessa forma para dizer ao SGBD que estamos querendo comparar uma coluna do tipo texto. Para coluna do tipo numérico, não há necessidade.

```
mysql> select n_numeclien, c_codiclien, c_razaclien
         from comclien
        where c_codiclien = '0001';

+-------------+-------------+------------------------+
| n_numeclien | c_codiclien | c_razaclien            |
+-------------+-------------+------------------------+
|           1 | 0001        | AARONSON FURNITURE LTD |
+-------------+-------------+------------------------+
1 row in set (0.00 sec)
```

E se quiséssemos o contrário? Todos os clientes que sejam diferentes de `'0001'`? Faríamos uma consulta utilizando o operador do MySQL que significa diferente: `<>`. Ficaria assim:

```
mysql> select n_numeclien, c_codiclien, c_razaclien
        from comclien
       where c_codiclien <> '0001';

+-------------+-------------+------------------------+
| n_numeclien | c_codiclien | c_razaclien            |
+-------------+-------------+------------------------+
```

```
|           2 | 0002        | LITTLER  LTDA          |
|           3 | 0003        | KELSEY  NEIGHBOURHOOD  |
|           4 | 0004        | GREAT AMERICAN MUSIC   |
|           5 | 0005        | LIFE PLAN COUNSELLING  |
|           6 | 0006        | PRACTI-PLAN LTDA       |
|           7 | 0007        | SPORTSWEST LTDA        |
|           8 | 0008        | HUGHES MARKETS LTDA    |
|           9 | 0009        | AUTO WORKS LTDA        |
|          10 | 00010       | DAHLKEMPER  LTDA       |
+-------------+-------------+------------------------+
9 rows in set (0.00 sec)
```

Observe que ele trouxe todos os clientes, exceto aquele cujo `c_codiclien` é igual a `'0001'`.

Além dos operadores de comparação = e <>, temos os seguintes:

- > : maior;
- < : menor;
- >=: maior e igual;
- <=: menor e igual.

Em vez de utilizarmos o = para comparar uma `string`, também podemos utilizar o `like`. Ele também é usado para isso e, excepcionalmente, para quando queremos consultar uma e só conhecemos uma parte dela. Por exemplo, se quisermos retornar todos os clientes que se iniciam com a letra B, montaríamos nossa consulta da seguinte maneira:

```
mysql> select n_numeclien, c_codiclien, c_razaclien
         from comclien
        where c_razaclien like 'L%';

+-------------+-------------+-----------------------+
| n_numeclien | c_codiclien | c_razaclien           |
+-------------+-------------+-----------------------+
|           5 | 0005        | LIFE PLAN COUNSELLING |
|           2 | 0002        | LITTLER  LTDA         |
```

```
+-------------+-------------+----------------------+
2 rows in set (0.01 sec)
```

O símbolo de % (porcento) é um curinga no SQL. Quando não sabemos uma parte da *string*, podemos utilizá-lo no início, no meio ou no fim dela.

Distinct()

E se fizéssemos uma lista de todos os clientes que compraram algo? Se fosse apenas a consulta:

```
mysql> select n_numeclien from comvenda;
```

Isso retornaria lista de clientes e de vendas e, se o cliente possuir mais de uma venda, apareceria repetido no resultado. Para não selecionar um registro igual ao outro, utilizamos o `DISTINCT`. Com o seguinte código, teremos a lista de clientes que fizeram ao menos uma compra e sem nenhuma repetição.

```
mysql> select distinct n_numeclien
             from comvenda;
+-------------+
| n_numeclien |
+-------------+
|           1 |
|           2 |
|           3 |
|           4 |
|           5 |
|           6 |
|           7 |
|           8 |
|           9 |
+-------------+
9 rows in set (0.00 sec)
```

No arquivo que disponibilizei para inserir os registros, o cliente com o `n_numeclien` igual a 10 não fez nenhuma compra e outros fizeram mais de uma. Mais à frente, vou mostrar como podemos contar as vendas de cada um.

5.2 *Subquery* ou subconsulta

As subconsultas são alternativas para as `joins`, que vamos ver logo a seguir. Utilizando-as, conseguimos ter um `select` dentro de outro `select` para nos ajudar a recuperar registros que estão referenciados em outras tabelas.

Antes de demonstrar o uso da *subquery*, vamos aprender a utilização das cláusulas `in`, `not in`, `exists` e `not exists`, pois precisaremos delas para criar verificações para nossas consultas.

Cláusulas in e not in

Até agora, utilizamos os sinais =, <>, >= e <== para as condições das consultas. Para fazermos comparações com *n* valores, não conseguiríamos fazer com esses que aprendemos até agora, pois eles aceitam apenas um valor para a comparação. Para exemplificarmos, vamos escrever uma consulta para retornar simultaneamente os clientes que possuem `n_numeclien` igual a 1 e 2.

```
mysql> select c_codiclien, c_razaclien
         from comclien
        where n_numeclien = 1,2;

ERROR 1241 (21000): Operand should contain 1 column(s)
```

Ops! Temos um erro. Apenas utilizaremos o sinal de = quando a comparação for só com um valor. Já as cláusulas `in` e `not in` surgem para fornecer apoio quando queremos testar um ou mais. Vamos fazer a mesma consulta utilizando o `in`. Lembre-se de colocar os valores entre parênteses e separados por vírgula. Se forem valores do tipo *string*, será entre aspas simples.

```
mysql> select c_codiclien, c_razaclien
         from comclien
        where n_numeclien in (1,2);

+-------------+------------------------+
| c_codiclien | c_razaclien            |
+-------------+------------------------+
| 0001        | AARONSON FURNITURE LTD |
```

```
| 0002         | LITTLER  LTDA           |
+-------------+------------------------+
2 rows in set (0.63 sec)
```

Ou podíamos consultar clientes que possuem o `n_nnumeclien` diferente de 1 e 2. Nesta ocasião, devemos utilizar o `not in`. Vamos ao código.

```
mysql> select c_codiclien, c_razaclien
         from comclien
        where n_numeclien not in (1,2);

+-------------+-----------------------+
| c_codiclien | c_razaclien           |
+-------------+-----------------------+
| 0003        | KELSEY  NEIGHBOURHOOD |
| 0004        | GREAT AMERICAN MUSIC  |
| 0005        | LIFE PLAN COUNSELLING |
| 0006        | PRACTI-PLAN LTDA      |
| 0007        | SPORTSWEST LTDA       |
| 0008        | HUGHES MARKETS LTDA   |
| 0009        | AUTO WORKS LTDA       |
| 00010       | DAHLKEMPER  LTDA      |
+-------------+-----------------------+
8 rows in set (0.00 sec)
```

Nas duas últimas consultas, nós sabíamos os números dos clientes que queríamos ou não consultar. Entretanto, em nosso projeto, surgiu a necessidade de criar uma para retornar a razão social dos clientes que possuem registro na tabela `comvenda`. Para esta situação, vamos utilizar uma subconsulta. A principal retornará a razão social do cliente e vai comparar o `n_numeclien` que será retornado pela subconsulta. Esta, por sua vez, retornará todos `n_nnumeclien` da tabela `comvenda`. Vamos utilizar a cláusula `in` e a subconsulta entre parênteses. Vamos ao código.

```
mysql> select c_razaclien
         from comclien
        where n_numeclien in  (select n_numeclien
                                 from comvenda
                                where n_numeclien);
```

```
+------------------------+
| c_razaclien            |
+------------------------+
| AARONSON FURNITURE LTD |
| AUTO WORKS LTDA        |
| GREAT AMERICAN MUSIC   |
| HUGHES MARKETS LTDA    |
| KELSEY  NEIGHBOURHOOD  |
| LIFE PLAN COUNSELLING  |
| LITTLER  LTDA          |
| PRACTI-PLAN LTDA       |
| SPORTSWEST LTDA        |
+------------------------+
9 rows in set (0.00 sec)
```

Utilizando a mesma situação, vamos buscar os clientes que ainda não fizeram nenhuma venda. Para isso, utilizaremos o `not in`. Você verá que o único cliente que ainda não possui registro de venda retornará, pois a consulta principal vai consultar todos os registros que não possuem o `n_nnumeclien` na tabela `comvenda`.

```
mysql> select c_razaclien
         from comclien
        where n_numeclien not in (select n_numeclien
                                     from comvenda);
+------------------+
| c_razaclien      |
+------------------+
| DAHLKEMPER  LTDA |
+------------------+
1 row in set (0.01 sec)
```

Você utilizará bastante as subconsultas em diversos cenários que surgirão em seu dia a dia. Ainda podemos ter uma com a característica de um campo da tabela, que retornará uma ou mais colunas de lugares diferentes. Exemplificando: vamos supor que, em nosso sistema, surgiu a necessidade de desenvolver uma consulta para retornar o código das vendas e a razão social dos respectivos clientes que as fizeram.

A consulta principal será um `select` na tabela `comvenda` junto com uma subconsulta. Esta terá uma vírgula separando-a do primeiro campo e o `n_nnumeclien` sendo passado da consulta principal, para realizar a comparação e buscar a razão social do respectivo cliente. Vamos ao código.

```
mysql> select c_codivenda Cod_Venda,
        (select c_razaclien
         from comclien
         where n_numeclien = comvenda.n_numeclien) Nome_Cliente
from comvenda;

+-------------+------------------------+
| Cod_Venda   | Nome_Cliente           |
+-------------+------------------------+
| 1           | AARONSON FURNITURE LTD |
| 2           | LITTLER  LTDA          |
| 3           | KELSEY  NEIGHBOURHOOD  |
| 4           | GREAT AMERICAN MUSIC
| 5           | LIFE PLAN COUNSELLING  |
| 6           | PRACTI-PLAN LTDA       |
| 7           | SPORTSWEST LTDA        |
| 8           | HUGHES MARKETS LTDA    |
| 9           | AUTO WORKS LTDA        |
| 10          | AARONSON FURNITURE LTD |
| 11          | AARONSON FURNITURE LTD |
| 12          | LITTLER  LTDA          |
| 13          | KELSEY  NEIGHBOURHOOD  |
| 14          | KELSEY  NEIGHBOURHOOD  |
| 15          | LIFE PLAN COUNSELLING  |
| 16          | PRACTI-PLAN LTDA       |
| 17          | SPORTSWEST LTDA        |
| 18          | HUGHES MARKETS LTDA    |
| 19          | AUTO WORKS LTDA        |
| 20          | AUTO WORKS LTDA        |
+-------------+------------------------+
20 rows in set (0.00 sec)
```

Essa maneira não é muito usada, porque há perda de performance e o código não fica legal. Por isso, aprenderemos a fazer *JOINS*: a forma correta

para retornamos valores de uma ou mais tabelas em um único select.

Criação de alias (apelidos das tabelas)

Observe o nosso último código. No cabeçalho do resultado, em vez de retornar os nomes das colunas, apareceram os que colocamos na frente das que estamos consultando. Ou seja, criamos apelidos. Você pode fazer isso em qualquer coluna em uma consulta. Em vez de mostrar seu nome no resultado, você pode exibir o título que quiser. Dizemos que estamos apelidando as colunas e isso é chamado de *alias*.

Para exemplificar, vamos consultar a c_codiclien e a c_nomeclien, colocando os alias *CODIGO* e *CLIENTE* respectivamente. Vamos ao código:

```
mysql> select c_codiclien CODIGO, c_nomeclien CLIENTE
         from comclien
        where n_numeclien not in(1,2,3,4);

+-------------+----------------------+
|   CODIGO    |     CLIENTE          |
+-------------+----------------------+
| 0005        | LIFE PLAN COUNSELLING |
| 0006        | PRACTI-PLAN          |
| 0007        | SPORTSWEST           |
| 0008        | HUGHES MARKETS       |
| 0009        | AUTO WORKS           |
| 00010       | DAHLKEMPER           |
+-------------+----------------------+
6 rows in set (0.00 sec)
```

Utilizamos os alias quando temos muitas colunas com nomes iguais, que estão retornando algum nome diferente ou que não faça sentido para quem você apresentará o retorno da consulta. Neste último caso, temos como exemplo o nome de alguma função, como veremos no capítulo 6. Já que teremos funções que serão longas, não será legal apresentar um relatório para um cliente mostrando seu nome em vez do que a coluna representa.

Por exemplo: pegaremos a consulta onde tivemos uma subconsulta fazendo o papel de uma coluna. Se nós não tivéssemos utilizado um alias, o

resultado de seu cabeçalho seria o que está escrito na subconsulta. Vamos ao código.

```
mysql> select c_codivenda,
             (select c_razaclien
                from comclien
               where n_numeclien = comvenda.n_numeclien)
        from comvenda;
+-------------+----------------------------------------------+
| c_codivenda | (select c_razaclien
                from comclien
                   where n_numeclien = comvenda.n_numeclien)  |
+-------------+----------------------------------------------+
| 1           | AARONSON FURNITURE LTD                       |
| 2           | LITTLER  LTDA                                |
| 3           | KELSEY  NEIGHBOURHOOD                        |
| 4           | GREAT AMERICAN MUSIC                         |
| 5           | LIFE PLAN COUNSELLING                        |
| 6           | PRACTI-PLAN LTDA                             |
| 7           | SPORTSWEST LTDA                              |
| 8           | HUGHES MARKETS LTDA                          |
| 9           | AUTO WORKS LTDA                              |
| 10          | AARONSON FURNITURE LTD                       |
| 11          | AARONSON FURNITURE LTD                       |
| 12          | LITTLER  LTDA                                |
| 13          | KELSEY  NEIGHBOURHOOD                        |
| 14          | KELSEY  NEIGHBOURHOOD                        |
| 15          | LIFE PLAN COUNSELLING                        |
| 16          | PRACTI-PLAN LTDA                             |
| 17          | SPORTSWEST LTDA                              |
| 18          | HUGHES MARKETS LTDA                          |
| 19          | AUTO WORKS LTDA                              |
| 20          | AUTO WORKS LTDA                              |
+-------------+----------------------------------------------+
20 rows in set (0.02 sec)
```

Observe agora ao utilizar o alias:

```
mysql> select c_codivenda Cod_Venda,
         (select c_razaclien
```

```
            from comclien
           where n_numeclien = comvenda.n_numeclien) Nome_Cliente
        from comvenda;

+-------------+------------------------+
| Cod_Venda   | Nome_Cliente           |
+-------------+------------------------+
| 1           | AARONSON FURNITURE LTD |
| 2           | LITTLER  LTDA          |
| 3           | KELSEY  NEIGHBOURHOOD  |
| 4           | GREAT AMERICAN MUSIC   |
| 5           | LIFE PLAN COUNSELLING  |
| 6           | PRACTI-PLAN LTDA       |
| 7           | SPORTSWEST LTDA        |
| 8           | HUGHES MARKETS LTDA    |
| 9           | AUTO WORKS LTDA        |
| 10          | AARONSON FURNITURE LTD |
| 11          | AARONSON FURNITURE LTD |
| 12          | LITTLER  LTDA          |
| 13          | KELSEY  NEIGHBOURHOOD  |
| 14          | KELSEY  NEIGHBOURHOOD  |
| 15          | LIFE PLAN COUNSELLING  |
| 16          | PRACTI-PLAN LTDA       |
| 17          | SPORTSWEST LTDA        |
| 18          | HUGHES MARKETS LTDA    |
| 19          | AUTO WORKS LTDA        |
| 20          | AUTO WORKS LTDA        |
+-------------+------------------------+
20 rows in set (0.00 sec)
```

Portanto, se você for mostrar para um usuário esse resultado, utilize o alias, pois, muito provavelmente, ele não saberá o que significa a sintaxe de um `select`.

5.3 Traga informação de várias tabelas com Joins

Até agora, selecionamos dados de apenas uma tabela. Ao fazer um relatório, as informações possivelmente estarão em várias delas. Para fazer um consulta em mais de uma, nós utilizamos os chamados `JOINs`. Há sintaxes diferentes para escrevê-lo. Utilizarei a mais comum e mais simples. No repositório, há o arquivo `consultas.sql`, que contém várias outras consultas para você estudar e utilizar como exemplo.

Nos bancos de dados relacionais, ao você consultar duas tabelas que possuem algum tipo de relacionamento, você deve especificar de qual tabela são esses campos. Vamos pegar como exemplo as tabelas de vendas e de clientes. Temos uma coluna que é igual entre elas: a chave primária da tabela de cliente e o `n_numeclien`. Com isso, temos `comvenda.n_numeclien = comclien.n_numeclien`. Veja que, além de fazer a igualdade entre as colunas, deve-se especificar também à qual tabela pertence cada campo.

Entendemos o funcionamento do `JOIN`, então agora podemos fazer uma extração de dados, relacionando as vendas com os clientes. Ao retirar um relatório de um sistema com muitos registros, eles devem estar organizados por alguma sequência, seja esta ordenada pelo código ou pelo nome do cliente, uma vez que fica estranho e ruim de ler um relatório que não esteja organizado. No SQL, também temos um comando para ordenar as consultas. Para isso, temos o `order by`. Ordenando pela razão social do cliente, o nosso código ficará da seguinte maneira:

```
mysql> select c_codiclien, c_razaclien, c_codivenda Cod_Venda
         from comvenda, comclien
        where comvenda.n_numeclien = comclien.n_numeclien
        order by c_razaclien;

+-------------+------------------------+-----------+
| c_codiclien | c_razaclien            | Cod_Venda |
+-------------+------------------------+-----------+
| 0001        | AARONSON FURNITURE LTD | 10        |
| 0001        | AARONSON FURNITURE LTD | 1         |
| 0001        | AARONSON FURNITURE LTD | 11        |
```

```
| 0009         | AUTO WORKS LTDA         | 20          |
| 0009         | AUTO WORKS LTDA         | 19          |
| 0009         | AUTO WORKS LTDA         | 9           |
| 0004         | GREAT AMERICAN MUSIC    | 4           |
| 0008         | HUGHES MARKETS LTDA     | 18          |
| 0008         | HUGHES MARKETS LTDA     | 8           |
| 0003         | KELSEY  NEIGHBOURHOOD   | 13          |
| 0003         | KELSEY  NEIGHBOURHOOD   | 3           |
| 0003         | KELSEY  NEIGHBOURHOOD   | 14          |
| 0005         | LIFE PLAN COUNSELLING   | 5           |
| 0005         | LIFE PLAN COUNSELLING   | 15          |
| 0002         | LITTLER  LTDA           | 12          |
| 0002         | LITTLER  LTDA           | 2           |
| 0006         | PRACTI-PLAN LTDA        | 16          |
| 0006         | PRACTI-PLAN LTDA        | 6           |
| 0007         | SPORTSWEST LTDA         | 7           |
| 0007         | SPORTSWEST LTDA         | 17          |
+-------------+------------------------+-----------+
20 rows in set (0.01 sec)
```

A maneira mais formal de escrever uma consulta com JOIN é como está apresentado a seguir. Porém, não é a mais comum e utilizada no dia a dia, pois o código fica um pouco mais complexo. Há uma grande discussão sobre desempenho das consultas na maneira como é escrita. Não entrarei no mérito desta. Durante o livro, utilizarei a sintaxe mais popular, que é a que apresentei anteriormente.

```
mysql> select c_codiclien codigo, c_razaclien razao_social,
       c_codivenda codi_venda
         from comvenda
         join comclien on
         comvenda.n_numeclien  = comclien.n_numeclien
        order by c_razaclien;
```

> **Atenção:** ao utilizar várias tabelas para fazer uma consulta, você deve sempre fazer a igualdade entre as que possuem `constraint`, pois, caso contrário, o SGBD se perderá e retornará os dados duplicados.

5.4 Select em: create table, insert, update e delete

Aprendemos a criar tabelas e a inserir e deletar registros no banco de dados por meio de comandos adequados. Porém, agora que aprendemos a realizar consultas, podemos utilizar o `select` para nos auxiliar nessas operações. Algo que pode ser muito útil em nosso dia a a dia, em que buscamos o máximo de produtividade.

Criando tabelas por meio de select

Surgiu a necessidade de criarmos uma tabela chamada `comclien_bkp` com a mesma estrutura e dados da `comclien`, onde o `c_estaclien` seja igual a `'SP'`. Podemos realizar algumas operações com esses registros e, por segurança, não usaremos os dados da tabela original.

Essa situação, na qual você precisa isolar alguns registros utilizando algum tipo de filtro para trabalhar com eles sem afetar a tabela que está em produção, é muito comum de encontrar. Em nosso cenário, este filtro será o `c_estaclien` igual a `'SP'`. Se você ainda não inseriu os registros, baixe o arquivo `popula_banco.sql` do repositório e aplique-os em seu banco.

```
mysql> create table comclien_bkp as(
         select *
           from comclien
          where c_estaclien = 'SP');

Query OK, 3 rows affected (0.70 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

Inserindo registros por meio de select

Constantemente, surge a necessidade de inserir registros em alguma ta-

bela, a fim de realizar algum processo no banco de dados. Às vezes, já temos esses dados em outra tabela e, com isso, em vez de criarmos scripts para inseri-los, nós podemos utilizar um `select` para buscar o que temos e colocar em nossa nova tabela.

Em nosso projeto, apareceu a necessidade da criação uma tabela para agenda telefônica. Ela terá como base alguns campos da tabela de clientes e todos eles serão cadastrados nela também. Veremos como esse processo será feito automaticamente no capítulo 7, mas para iniciarmos, vamos inserir os clientes que possuímos atualmente por meio de um `select`. Observe também que criarei um campo `n_nnumeclien` que será a *foreign key* da tabela de clientes, porém vou criá-la sem a obrigatoriedade de ser preenchida, uma vez que podemos ter contatos que não serão necessariamente um cliente. Isso é comum em sistemas.

```
mysql> create table comcontato(
          n_numecontato int not null auto_increment,
          c_nomecontato varchar(200),
          c_fonecontato varchar(30),
          c_cidacontato varchar(200),
          c_estacontato varchar(2),
          n_numeclien   int,
          primary key(n_numecontato));

Query OK, 0 rows affected (2.07 sec)
```

Agora vamos popular as colunas da nossa tabela `comcontato` com essas informações que temos da tabela `comclien`.

```
mysql> insert into comcontato(
       select n_numeclien,
              c_nomeclien,
              c_foneclien,
              c_cidaclien,
              c_estaclien,
              n_numeclien
        from comclien);

Query OK, 10 rows affected (0.16 sec)
Records: 10  Duplicates: 0  Warnings: 0
```

Para visualizar os registros da nossa tabela, faça um `select` simples para listá-los.

```
mysql> select * from comcontato;

+---------------+----------------------+---------------
| n_numecontato | c_nomecontato        | c_fonecontato
+---------------+----------------------+---------------
|             1 | AARONSON FURNITURE   | (21) 8167-6584
|             2 | LITTLER              | (27) 7990-9502
|             3 | KELSEY  NEIGHBOURHOOD | (11) 4206-9703
|             4 | GREAT AMERICAN MUSIC | (75) 7815-7801
|             5 | LIFE PLAN COUNSELLING | (17) 4038-9355
|             6 | PRACTI-PLAN          | (28) 2267-6159
|             7 | SPORTSWEST           | (61) 4094-7184
|             8 | HUGHES MARKETS       | (21) 7984-9809
|             9 | AUTO WORKS           | (21) 8548-5555
|            10 | DAHLKEMPER           | (11) 4519-7670
+---------------+----------------------+---------------
+-------------------------+---------------+-------------+
| c_cidacontato           | c_estacontato | n_numeclien |
+-------------------------+---------------+-------------+
| QUEIMADOS               | RJ            |           1 |
| SERRA                   | ES            |           2 |
| BRAGANÇA PAULISTA       | SP            |           3 |
| SANTO ANTÔNIO DE JESUS  | BA            |           4 |
| BEBEDOURO               | SP            |           5 |
| CACHOEIRO DE ITAPEMIRI  | ES            |           6 |
| TAGUATINGA              | DF            |           7 |
| RIO DE JANEIRO          | RJ            |           8 |
| RIO DE JANEIRO          | RJ            |           9 |
| SÃO PAULO               | SP            |          10 |
+-------------------------+---------------+-------------+
10 rows in set (0.00 sec)
```

Alterando registros por meio de select

Neste momento, descobrimos que os contatos dos cliente que estão na tabela `comclien_bkp`, na verdade, possuem o contato em outra ci-

dade e estado, diferente dos dados que estão na comclien. Ainda utilizando um select, faremos um update nos campos c_cidacontato e c_estacontato, buscando os registros da tabela comclien_bkp e alterando a comcontato.

```
mysql> update comcontato set c_cidacontato = 'LONDRINA',
                             c_estacontato = 'PR'
        where n_numeclien in ( select n_numeclien
                                 from comclien_bkp);

Query OK, 3 rows affected (0.31 sec)
Rows matched: 3  Changed: 3  Warnings: 0
```

Deletando registros por meio de select

Como eu sempre digo: você deve estar preparado para realizar mudanças em seu projeto. Para isso, devemos conhecer o que podemos fazer com o que o MySQL nos fornece. Por exemplo, agora temos a necessidade de deletar todos os registros da tabela comcontato, pois os contatos não possuem registros na tabela comvenda; ou seja, aqueles que não possuem nenhuma venda.

```
mysql> delete from comcontato
       where n_numeclien not in (select n_numeclien
                                   from comvenda );

Query OK, 1 rows affected (0.09 sec)

mysql> commit;
Query OK, 0 rows affected (0.00 sec)
```

Agora, se consultarmos a tabela comcontato, não veremos o contato que não possuía nenhum registro na comvenda.

```
mysql> select * from comcontato;
+---------------+-----------------------+----------------
| n_numecontato | c_nomecontato         | c_fonecontato
+---------------+-----------------------+----------------
|             1 | AARONSON FURNITURE    | (21) 8167-6584
|             2 | LITTLER               | (27) 7990-9502
```

```
|              3 | KELSEY  NEIGHBOURHOOD | (11) 4206-9703
|              4 | GREAT AMERICAN MUSIC  | (75) 7815-7801
|              5 | LIFE PLAN COUNSELLING | (17) 4038-9355
|              6 | PRACTI-PLAN           | (28) 2267-6159
|              7 | SPORTSWEST            | (61) 4094-7184
|              8 | HUGHES MARKETS        | (21) 7984-9809
|              9 | AUTO WORKS            | (21) 8548-5555
+---------------+-----------------------+-----------------

+-------------------------+
| c_cidacontato           |
+-------------------------+
| QUEIMADOS               |
| SERRA                   |
| LONDRINA                |
| SANTO ANTÔNIO DE JESUS  |
| LONDRINA                |
| CACHOEIRO DE ITAPEMIRI  |
| TAGUATINGA              |
| RIO DE JANEIRO          |
| RIO DE JANEIRO          |
+-------------------------+

+---------------+---------------+-------------+
| n_numecontato | c_estacontato | n_numeclien |
+---------------+---------------+-------------+
|             1 | RJ            |           1 |
|             2 | ES            |           2 |
|             3 | PR            |           3 |
|             4 | BA            |           4 |
|             5 | PR            |           5 |
|             6 | ES            |           6 |
|             7 | DF            |           7 |
|             8 | RJ            |           8 |
|             9 | RJ            |           9 |
+---------------+---------------+-------------+

9 rows in set (0.00 sec)
```

Utilize o máximo que seu SGBD tem a oferecer. Não perca tempo montando scripts desnecessários se você tem de usar dados que já estão no banco.

Todos esses comandos, além de estarem em nosso repositório, estão no final do livro reunidos no guia de consulta rápida, cada um com sua sintaxe e seu significado.

Neste capítulo, aprendemos a consultar os registro do nosso SGBD, algumas consultas simples e outras um pouco mais complexas.

A dica que deixo é praticar bastante. Como todas as linguagens de programação, o SQL também é uma questão de prática. Com o tempo, você não precisará pensar qual comando deve utilizar, apenas vai fazê-lo. Agora, no próximo capítulo, vamos aprender algumas funções para usarmos em nossas consultas.

Capítulo 6

Consultas com funções

"A vida é como um jardim. Momentos perfeitos podem ser desfrutados, mas não preservados, exceto na memória."
– Leonard Nimoy

6.1 Funções

No MySQL, existem várias funções nativas que nos possibilitam fazer diversas operações, dentre elas: realizar cálculos, manipular strings, trabalhar com datas, realizar opções lógicas, extrair informações dos nossos registros etc. Elas estão divididas nos seguintes tipo: numéricas, lógica, manipulação de string e funções de data e hora.

Explicarei as funções principais e quais você mais utilizará em seu dia a dia. Exemplificarei conforme surgir a necessidade de cada uma. Vale lembrar

que essas funções e operadores podem ser utilizados em qualquer cláusula `sql`, exceto na `from`.

6.2 Funções de agregação

As funções de agregação são responsáveis por agrupar vários valores e retornar somente um único para um determinado grupo. Por exemplo, se fizermos um `select` em todos registros da tabela de vendas com `join` com a tabela de clientes, vamos ter como resultado clientes repetidos. Veja:

```
mysql> select c_codiclien, c_razaclien
         from comvenda, comclien
        where comvenda.n_numeclien = comclien.n_numeclien
        order by c_razaclien;

+-------------+------------------------+
| c_codiclien | c_razaclien            |
+-------------+------------------------+
| 0001        | AARONSON FURNITURE LTD |
| 0001        | AARONSON FURNITURE LTD |
| 0001        | AARONSON FURNITURE LTD |
| 0009        | AUTO WORKS LTDA        |
| 0009        | AUTO WORKS LTDA        |
| 0009        | AUTO WORKS LTDA        |
| 0004        | GREAT AMERICAN MUSIC   |
| 0008        | HUGHES MARKETS LTDA    |
| 0008        | HUGHES MARKETS LTDA    |
| 0003        | KELSEY  NEIGHBOURHOOD  |
| 0003        | KELSEY  NEIGHBOURHOOD  |
| 0003        | KELSEY  NEIGHBOURHOOD  |
| 0005        | LIFE PLAN COUNSELLING  |
| 0005        | LIFE PLAN COUNSELLING  |
| 0002        | LITTLER  LTDA          |
| 0002        | LITTLER  LTDA          |
| 0006        | PRACTI-PLAN LTDA       |
| 0006        | PRACTI-PLAN LTDA       |
| 0007        | SPORTSWEST LTDA        |
| 0007        | SPORTSWEST LTDA        |
```

```
+-------------+-----------------------+
20 rows in set (0.01 sec)
```

Alguns clientes repetem-se, pois existem aqueles que possuem mais de uma venda. Desta maneira, poderíamos utilizar uma função de agregação para retorná-los, evitando a repetição.

Group by

O comando SQL para fazer essa operação de agregação é o `group by`. Ele deverá ser utilizado logo após as cláusulas de condições `where` ou `and`, e antes do `order by`, se a sua consulta possuí-lo.

Vamos ao nosso código:

```
mysql> select c_codiclien, c_razaclien
        from comclien, comvenda
       where comvenda.n_numeclien = comclien.n_numeclien
       group by c_codiclien, c_razaclien
       order by c_razaclien;

+-------------+-----------------------+
| c_codiclien | c_razaclien           |
+-------------+-----------------------+
| 0001        | AARONSON FURNITURE LTD |
| 0009        | AUTO WORKS LTDA       |
| 0004        | GREAT AMERICAN MUSIC  |
| 0008        | HUGHES MARKETS LTDA   |
| 0003        | KELSEY  NEIGHBOURHOOD |
| 0005        | LIFE PLAN COUNSELLING |
| 0002        | LITTLER  LTDA         |
| 0006        | PRACTI-PLAN LTDA      |
| 0007        | SPORTSWEST LTDA       |
+-------------+-----------------------+
9 rows in set (0.00 sec)
```

O MySQL agrupou o código e a razão social, trazendo apenas um registro de cada. Porém, essa consulta poderia ser melhor se tivéssemos a quantidade de vendas de cliente. Podemos utilizar uma outra função de agregação chamada `count()` para contar os registros que estão agrupados. Ela só pode

ser utilizada na cláusula `select`, pois contará os registros da coluna que está sendo selecionada. Complementando o código anterior, teremos:

```
mysql> select c_codiclien, c_razaclien, count(n_numevenda) Qtde
        from comclien, comvenda
       where comvenda.n_numeclien = comclien.n_numeclien
       group by c_codiclien, c_razaclien
       order by c_razaclien;

+-------------+------------------------+------+
| c_codiclien | c_razaclien            | Qtde |
+-------------+------------------------+------+
| 0001        | AARONSON FURNITURE LTD |    3 |
| 0009        | AUTO WORKS LTDA        |    3 |
| 0004        | GREAT AMERICAN MUSIC   |    1 |
| 0008        | HUGHES MARKETS LTDA    |    2 |
| 0003        | KELSEY  NEIGHBOURHOOD  |    3 |
| 0005        | LIFE PLAN COUNSELLING  |    2 |
| 0002        | LITTLER  LTDA          |    2 |
| 0006        | PRACTI-PLAN LTDA       |    2 |
| 0007        | SPORTSWEST LTDA        |    2 |
+-------------+------------------------+------+
9 rows in set (0.00 sec)
```

O `count` pode ser usado apenas para contar a quantidade de registro em uma tabela. Vamos substituir a coluna que estava entre parênteses no exemplo anterior por `*` (asterisco), para contar todas as linhas da tabela de clientes.

```
mysql> select count(*)
         from comclien;

+----------+
| count(*) |
+----------+
|       10 |
+----------+
1 row in set (0.05 sec)
```

Having count()

Agora, em nosso projeto, temos a necessidade de fazer um relatório que traga como resultado os clientes que tiveram mais do que duas vendas. Para isso, utilizaremos a função `having count()`, que será a condição para o seu `count()`. Exemplificando, temos:

```
mysql> select c_razaclien, count(n_numevenda)
         from comclien, comvenda
        where comvenda.n_numeclien = comclien.n_numeclien
        group by c_razaclien
       having count(n_numevenda) > 2;

+------------------------+--------------------+
| c_razaclien            | count(n_numevenda) |
+------------------------+--------------------+
| AARONSON FURNITURE LTD |                  3 |
| AUTO WORKS LTDA        |                  3 |
| KELSEY  NEIGHBOURHOOD  |                  3 |
+------------------------+--------------------+
3 rows in set (0.00 sec)
```

Percebeu que utilizamos todos os comando SQL que aprendemos em uma única consulta? Sempre um complementará o outro. Com o tempo, escrevê-los ficará natural e automático, conforme sua necessidade. Tudo é uma questão de prática, como qualquer linguagem de programação.

max() e min()

Depois de uma consulta mais complexa que pode ter feito você desistir de estudar banco de dados, vamos para uma que é um pouco mais simples. Muitas vezes, por *n* motivos, surge a necessidade de retornar o maior ou menor registro de uma tabela. Fazemos isso com as funções `MAX()` e `MIN()`, respectivamente. Nos parênteses deverá ir a coluna que você deseja recuperar. São funções simples do SQL que são de grande utilidade.

Se quisermos recuperar o valor da maior venda, nossa consulta seria:

```
mysql> select max(n_totavenda) maior_venda
         from comvenda;
```

```
+-------------+
| maior_venda |
+-------------+
|    25141.02 |
+-------------+
1 row in set (0.00 sec)
```

Já para a menor:

```
mysql> select min(n_totavenda) menor_venda, max(n_totavenda)
        maior_venda from comvenda;

+-------------+
| menor_venda |
+-------------+
|     4650.64 |
+-------------+
```

Ou ainda podemos retornar os dois valores ao mesmo tempo, da seguinte maneira:

```
mysql> select min(n_totavenda) menor_venda, max(n_totavenda)
        maior_venda from comvenda;

+-------------+-------------+
| menor_venda | maior_venda |
+-------------+-------------+
|     4650.64 |    25141.02 |
+-------------+-------------+
1 row in set (0.00 sec)
```

Sum()

Temos os valores das vendas, mas é óbvio que, em algum momento, teremos que consultar seu total. No MySQL, podemos somar todos os valores de uma coluna utilizando a função `sum()`. Como exemplo, vamos somar os valores individualmente das colunas: `n_valovenda`, `n_descvenda` e `n_totavenda` no intervalo de *01/01/2015* a *31/01/2015*.

```
mysql> select sum(n_valovenda) valor_venda,
              sum(n_descvenda) descontos,
                sum(n_totavenda) total_venda
         from comvenda
        where d_datavenda between '2015-01-01' and '2015-01-01';

+-------------+-----------+-------------+
| valor_venda | descontos | total_venda |
+-------------+-----------+-------------+
|    75830.72 |      0.00 |    75830.72 |
+-------------+-----------+-------------+
1 row in set (0.00 sec)
```

Observe que utilizamos a condição `between`, que serve para verificar um intervalo entre duas variáveis, seja de datas ou numérico.

Avg()

Conseguimos extrair os valores das vendas, de sua quantidade, as maiores e as menores etc. Porém, e se quisermos saber sua média, para comparar períodos de datas? No MySQL temos o `avg()`, que busca a coluna cuja média você deseja saber e realiza o cálculo. Vamos exemplificar consultando o valor médio de todas as vendas:

```
mysql> select format(avg(n_totavenda),2)
         from comvenda;

+----------------------------+
| format(avg(n_totavenda),2) |
+----------------------------+
| 12,213.96                  |
+----------------------------+
1 row in set (0.00 sec)
```

6.3 Funções de string

As funções de string (caracteres) podem ser utilizadas para modificar os dados, no que diz respeito aos valores selecionados, como também na forma

como são apresentados. Ou ainda, modificá-los para uma validação. Temos uma variedade de funções que são muito úteis, e que nos ajudam a resolver problemas do dia a dia, como tento mostrar em cada exemplo a seguir.

substr() e length()

Agora, em nosso projeto, surgiu a necessidade de consultar os produtos que iniciam seu código com `'123'` e que possuem uma descrição com mais de 4 caracteres, pois foram cadastrados de maneira errada. Poderíamos buscar todos os produtos, colocá-los em uma planilha e no 'olhômetro' encontrar todos eles. Mas e se eu lhe disser que existem funções no SQL que podemos utilizar para fazer esse filtro? São duas: a função `SUBSTR()` e a `length()`.

Imaginem um cenário com uma tabela com mais de 1 milhão de registros. Precisamos ter ferramentas para nos auxiliar. Diferente das outras funções para as quais apenas passamos a coluna, para esta devemos também passar qual o intervalo de caracteres que queremos de um determinado campo.

Por exemplo, `substr(c_codiprodu,1,3) = '123'`. Com este comando, falamos para o SGBD que queremos os registros que possuem o código da posição 1 até a posição 3 com a sequência de caracteres `123`. Com a função `LENGTH()`, vamos contar quantos caracteres o código do produto tem.

```
mysql> select c_codiprodu, c_descprodu
           from comprodu
          where substr(c_codiprodu,1,3) = '123'
            and length(c_codiprodu) > 4;

+-------------+-------------+
| c_codiprodu | c_descprodu |
+-------------+-------------+
| 123131      | NOTEBOOK    |
| 123223      | SMARTPHONE  |
+-------------+-------------+
2 rows in set (0.03 sec)
```

Percebeu a importância de trabalhar com uma coluna de chave primária e uma para o código que aparecerá na tela? Você consegue trabalhar de diversas maneiras, pois, dependendo da regra de negócio, pode ocorrer de dois produtos terem o mesmo código. Além de poder alterar sem problema algum, as comparações sempre serão pela chave primária.

Utilizamos, no exemplo, o `substr()` e o `length()` para fazermos uma validação. Poderíamos ter utilizado para apresentar os valores. Vamos selecionar apenas os cinco primeiros caracteres do campo `c_razaclien` e contar quantos deles temos no código do cliente. Vamos ao código:

```
mysql> select substr(c_razaclien,1,5) Razao_Social,
              length(c_codiprodu)     Tamanho_Cod
         from comclien
        where n_numeclien = 1;

+-------------+--------------+
| Razao_Social | Tamanho_Cod |
+-------------+--------------+
| AARON        |           6 |
+-------------+--------------+
1 rows in set (0.00 sec)
```

Concat() e concat_ws()

Queremos agora listar os clientes concatenando a razão social e o telefone. Temos a função `concat()` que concatena dois ou mais campos. Deve-se apenas colocá-los entre parênteses, separados por vírgula.

```
mysql> select concat(c_razaforne,' - fone: ', c_foneforne)
         from comforne
        order by c_razaforne;

+--------------------------------------------------------+
| concat(c_razaforne,' - fone: ', c_foneforne)           |
+--------------------------------------------------------+
```

```
| DUN RITE LAWN MAINTENANCE LTDA - fone: (85) 7886-8837 |
| SEWFRO FABRICS LTDA - fone: (91) 5171-8483            |
| WISE SOLUTIONS LTDA - fone: (11) 5347-5838            |
+-------------------------------------------------------+
3 rows in set (0.04 sec)
```

Por alguma necessidade, precisamos fazer consultas e concatenar mais de um campo. O MySQl nos permite fazer isso através das funções `concat()` e `concat_ws()`. Com o `concat()` será para concatenar todos os campos da consulta sem especificar um separador entre os campos; já com o `concat_ws()` devemos dizer qual será o separador entre eles. Vamos aos exemplos:

```
mysql> select
       concat(c_codiclien,'  ',c_razaclien, '  ', c_nomeclien)
         from comclien
        where c_razaclien like 'GREA%';

+---------------------------------------------------------+
| concat(c_codiclien,'  ',c_razaclien, '  ', c_nomeclien) |
+---------------------------------------------------------+
| 0004  GREAT AMERICAN MUSIC  GREAT AMERICAN MUSIC        |
+---------------------------------------------------------+
1 row in set (0.00 sec)
```

Olhando para o resultado, observe que nós separamos os campos com duplo espaço. Poderíamos fazer isso utilizando algum caractere especial, como com ponto e vírgula (`;`):

```
mysql> select
       concat_ws(';',c_codiclien, c_razaclien, c_nomeclien)
         from comclien
        where c_razaclien like 'GREA%';

+------------------------------------------------------+
| concat_ws(';',c_codiclien, c_razaclien, c_nomeclien) |
+------------------------------------------------------+
| 0004;GREAT AMERICAN MUSIC;GREAT AMERICAN MUSIC       |
+------------------------------------------------------+
1 row in set (0.00 sec)
```

Observe que agora apenas declaramos qual o separador queríamos e o SGBD colocou-o entre os campos.

Lcase() e lower()

Se você fizer uma consulta em nosso banco, vai perceber que alguns registros estão em letras maiúsculas. Se você necessitar, em algum lugar de sua aplicação, dos registros em letras minúsculas, o MySQL também tem uma função para auxiliá-lo. Utilize o `lcase` ou o `lower` da seguinte maneira:

```
mysql> select lcase(c_razaclien)
         from comclien;

+------------------------+
| lcase(c_razaclien)     |
+------------------------+
| aaronson furniture ltd |
| auto works ltda        |
| dahlkemper  ltda       |
| great american music   |
| hughes markets ltda    |
| kelsey  neighbourhood  |
| life plan counselling  |
| littler  ltda          |
| practi-plan ltda       |
| sportswest ltda        |
+------------------------+
10 rows in set (0.00 sec)
```

Ucase()

Da mesma maneira que podemos retornar os registros de forma minúscula, podemos também de forma maiúscula. Utilize a função `ucase`. Vamos consultar:

```
mysql> select ucase('banco de dados mysql')
         from dual;

+-------------------------------+
```

```
| ucase('banco de dados mysql') |
+-------------------------------+
| BANCO DE DADOS MYSQL          |
+-------------------------------+
1 row in set (0.07 sec)
```

6.4 Funções de cálculos e operadores aritméticos

Funções de cálculos, como a descrição já diz, são utilizadas para realizar operações de cálculos. É de grande utilidade ter essas funções, pois, assim, não é preciso realizá-los usando apenas operadores comuns. Por exemplo, temos desde uma função para realizar o cálculo da raiz quadrada até uma que retorna a tangente de Pi!

Round()

Quando criamos a tabela, nós especificamos que os campos do tipo `float()` seriam limitados em duas casas decimais. Porém, você pode se deparar com outros banco de dados que não estão limitados e possuem campos com registros com mais de duas casas. Para esses casos, nós temos a função `round()`, utilizada para arredondar valores. Você pode especificar para quantas casas decimais quer arredondar. Vamos exemplificar utilizando novamente a tabela `dual`:

```
mysql> select round('213.142',2)
         from dual;

+--------------------+
| round('213.142',2) |
+--------------------+
|             213.14 |
+--------------------+
1 row in set (0.00 sec)
```

Ou ainda, outra alternativa para arredondar um valor é utilizando o `format`. Ele não faz um arredondamento, mas formata o valor para pare-

cer com apenas as casas decimais desejadas. Apenas substitua o `round` por `format`, pois além de formatá-las, ele formatará também todo o número.

```
mysql> select format('21123.142',2) from dual;

+-----------------------+
| format('21123.142',2) |
+-----------------------+
| 21,123.14             |
+-----------------------+
1 row in set (0.00 sec)
```

Truncate

Utilizamos o `round` para arredondar e o `format` para arredondar e formatar o número. Temos a opção de utilizar uma função que vai truncar as casas decimais, ou seja, omiti-las. Exemplificando:

```
mysql> select truncate(max(n_totavenda),0) maior_venda
         from comvenda;

+-------------+
| maior_venda |
+-------------+
|       25141 |
+-------------+
1 row in set (0.01 sec)
```

Dependendo da situação com que você está lidando, você poderá deixar alguma casa decimal. Basta substituir o número zero que coloquei pelo número de casas decimais que deseja truncar. Por exemplo:

```
mysql> select truncate(min(n_totavenda),1) menor_venda
         from comvenda;

+-------------+
| menor_venda |
+-------------+
|      4650.6 |
```

```
+-------------+
1 row in set (0.01 sec)
```

Sqrt()

Dependendo do projeto com que você está trabalhando, a necessidade de se obter a raiz quadrada para realizar alguma operação pode aparecer. Graças ao MySQL, não precisamos escrever mais que duas linhas de código para se obter a raiz quadrada de um número. Isso não é fantástico? Vamos ao código:

```
mysql> select sqrt(4);

+---------+
| sqrt(4) |
+---------+
|       2 |
+---------+
1 row in set (0.00 sec)
```

Pi, seno, cosseno e tangente

Outras funções interessantes de que o MySQL dispõe são as funções: seno, cosseno e tangente. Lembra-se delas? E do número Pi? Temos as calculadoras que nos fornecem esses valores ao apertar de um botão, mas, muitas vezes, é necessário a utilização desses cálculos no desenvolvimento de sistemas. Para facilitar nossas vidas, temos essas funções em apenas uma linha de código. Vamos ao código:

Para consultar o valor de Pi:

```
mysql> select pi();

+----------+
|   pi()   |
+----------+
| 3.141593 |
+----------+
1 row in set (0.00 sec)
```

Para consultar o valor de seno de Pi:

```
mysql> select sin(pi());

+------------------+
| round(sin(pi())) |
+- ----------------+
|               0  |
+------------------+
1 row in set (0.00 sec)
```

Para consultar o valor de cosseno de Pi:

```
mysql> select cos(pi());

+-----------+
| cos(pi()) |
+-----------+
|        -1 |
+-----------+
1 row in set (0.00 sec)
```

Para consultar o valor da tangente de Pi + 1:

```
mysql> select tan(pi()+1);

+-----------------+
|      tan(pi())  |
+-----------------+
| 1.5574077246549 |
+-----------------+
1 row in set (0.00 sec)
```

Caso você tente fazer uma operação que não é permitida pela função, o MySQL retornará um erro de sintaxe, como esse da próxima consulta de exemplo, na qual tentei retornar a tangente de `pi()+(a)`. Visto que é uma função numérica, não aceitará caracteres `string`.

```
mysql> select tan(pi()+(A));
ERROR 1064 (42000): You have an error in your SQL syntax; check
the manual that corresponds to your MySQL server version for the
right syntax to use near '' at line 1
```

> **Dica**: este erro aparecerá em qualquer consulta quando um erro de sintaxe ocorrer. Uma dica é copiar seu nome, `(ERROR 1064 (42000))`, e procurá-lo no Google com a ação que você estiver fazendo. Exemplo: *ERROR 1064 (42000) tan MySQL*. Por ser um erro genérico, ao pesquisar também o que você está tentando fazer, a chance de encontrar a resposta será maior.

6.5 Operadores aritméticos

Utilizamos os operadores aritméticos quando temos que realizar cálculos para os quais não possuímos uma função para a operação, ou se for necessário fazer cálculos de operações para os quais já exista uma função. A seguir, veja os operadores em sua sequência de prioridade:

- $*$: multiplicação;
- $/$: divisão;
- $+$: adição;
- $-$: subtração.

Para realizar os cálculos, devemos utilizá-los na cláusula `select`, como as demais funções. Exemplificando: vamos multiplicar a quantidade de um produto de uma venda por seu valor. Assim, teremos o valor total de um item.

```
mysql> select (n_qtdeivenda * n_valoivenda) multiplicação
         from comivenda
        where n_numeivenda = 4;

+-----------------+
| multiplicação   |
+-----------------+
|        41038.72 |
+-----------------+
1 row in set (0.00 sec)
```

Agora vamos somar todos os valores de produtos dos itens das vendas e dividir pelo número de itens vendidos.

```
mysql>  select truncate((sum(n_valoivenda) /
        count(n_numeivenda)),2) divisão
          from comivenda;
         a
+----------+
| divisão  |
+----------+
|  6855.35 |
+----------+
1 row in set (0.00 sec)
```

Utilizando o item de venda com o `n_numeivenda` igual a 4, vamos somar o valor do item com o valor do desconto:

```
mysql> select (n_valoivenda + n_descivenda) adição
         from comivenda
        where n_numeivenda = 4;

+----------+
| adição   |
+----------+
| 10259.68 |
+----------+
1 row in set (0.00 sec)
```

Fazendo o inverso, vamos subtrair o valor do desconto do item:

```
mysql> select (n_valoivenda - n_descivenda) subtração
         from comivenda
        where n_numeivenda = 4;

+-------------+
| subtração   |
+-------------+
|    10259.68 |
+-------------+
1 row in set (0.00 sec)
```

6.6 Funções de data

Trabalhar com data em banco de dados pode tornar-se um grande problema, principalmente pela questão do padrão de datas em alguns sistemas. É muito comum você conhecer desenvolvedores que já tiveram problemas ou estão tendo, porque cada linguagem de programação trata a data de maneira diferentes. Se você souber manipulá-la da maneira correta, você terá sucesso.

Para nos auxiliar, o MySQL fornece funções com as quais podemos manipulá-las juntamente com o tempo. Vale lembrar que o MySQL utiliza o padrão americano: YYYY-MM-DD (ano, mês e dia). Na sequência, mostro como utilizar uma função para selecionar as datas em outro padrão. Vamos conhecer algumas delas.

Para retornar a data, hora ou data/hora atual, existem algumas maneiras de se fazer:

- `CURDATE()`: para retornar a data atual, somente. Por exemplo:

```
mysql> select curdate();

+------------+
| curdate()  |
+------------+
| 2015-03-03 |
+------------+
1 row in set (0.02 sec)
```

- `now()`: para retornar a data e a hora atual. Por exemplo:

```
mysql>select now();

+---------------------+
| now()               |
+---------------------+
| 2015-03-03 13:03:11 |
+---------------------+
1 row in set (0.00 sec)
```

- `sysdate()`: igualmente ao `now()`, sua consulta retorna a data e a hora juntos. Por exemplo:

```
mysql>select sysdate();

+---------------------+
| sysdate()           |
+---------------------+
| 2015-03-03 13:03:11 |
+---------------------+
1 row in set (0.00 sec)
```

- `curtime()`: para retornar somente o horário atual. Por exemplo:

```
mysql> select curtime();

+-----------+
| curtime() |
+-----------+
| 12:56:36  |
+-----------+
1 row in set (0.00 sec)
```

Podemos também retornar o intervalo entre duas datas:

```
mysql> select datediff('2015-02-01 23:59:59','2015-01-01');

+----------------------------------------------+
| datediff('2015-02-01 23:59:59','2015-01-01') |
+----------------------------------------------+
|                                           31 |
+----------------------------------------------+
1 row in set (0.00 sec)
```

E adicionar dias a uma data:

```
mysql>select date_add('2013-01-01', interval 31 day);

+-----------------------------------------+
| date_add('2013-01-01', interval 31 day) |
+-----------------------------------------+
| 2013-02-01                              |
+-----------------------------------------+
1 row in set (0.00 sec)
```

A função de selecionar o nome do dia da semana é muito útil. Você retornará o nome do dia da semana em vez de apenas a data com números, na tela para o seu usuário.

```
mysql> select dayname('2015-01-01');

+-----------------------+
| dayname('2015-01-01') |
+-----------------------+
| thursday              |
+-----------------------+
1 row in set (0.00 sec)
```

Para retornar o dia do mês:

```
mysql> select dayofmonth('2015-01-01');

+--------------------------+
| dayofmonth('2015-01-01') |
+--------------------------+
|                        1 |
+--------------------------+
1 row in set (0.02 sec)
```

Extrair o ano de uma data:

```
mysql> select extract(year from '2015-01-01');

+---------------------------------+
| extract(year from '2015-01-01') |
+---------------------------------+
|                            2015 |
+---------------------------------+
1 row in set (0.00 sec)
```

Extrair o último dia do mês:

```
mysql> select last_day('2015-02-01');

+------------------------+
```

```
| last_day('2015-02-01') |
+------------------------+
| 2015-02-28             |
+------------------------+
1 row in set (0.00 sec)
```

Formatando datas

Podemos fazer algumas formatações para apresentar as datas nas consultas.

Um padrão de data que utilizaremos bastante é o **EUR** (DD.MM.YYYY), pois ele é parecido com o nosso. Porém, lembre-se que é apenas para formatarmos durantes nossas consultas. Veja o exemplo onde vamos formatar a data '2015-01-10':

```
mysql> select date_format('2015-01-10',get_format(date,'EUR'));

+--------------------------------------------------+
| date_format('2015-01-01',get_format(date,'EUR')) |
+--------------------------------------------------+
| 10.01.2015                                       |
+--------------------------------------------------+
1 row in set (0.00 sec)
```

Você pode deparar-se com situações nas quais, por exemplo, tem de fazer uma migração de dados para o seu banco MySQL, mas os campos de data deste outro estão em um formato diferente e como tipo texto. Com isso, você terá de converter o campo para tipo data e para um formato compatível com o seu banco. Para converter de texto para data, utilizaremos a função `str_to_date` e, em seguida, passaremos para o nosso formato. Veja o exemplo:

```
mysql> select str_to_date('01.01.2015',get_format(date,'USA'));
+--------------------------------------------------+
| str_to_date('01.01.2015',get_format(date,'USA')) |
+--------------------------------------------------+
| 2015-01-01                                       |
+--------------------------------------------------+
1 row in set (0.00 sec)
```

Não se preocupe em aprender todas as funções de uma só vez. Você assimilará conforme forem surgindo suas necessidades, pois esta é a melhor forma de se aprender a programar. Neste capítulo, aprendemos as funções mais utilizadas, o que já lhe possibilita fazer *n* tipos de consultas. Agora vamos aprender algo um pouco mais complexo e deixar o processamento das informações no nosso potente SGBD.

Capítulo 7

Deixar o banco processar: procedures e functions

“Notícias ruins não são como os vinhos. Não melhoram com a idade.”
– Colin Powell

Os SGBDs são uma poderosa ferramenta de processamento de dados. Além de sua capacidade de gerenciar uma grande quantidade de dados, ele também é o mais rápido em comparação com aplicações escritas em outras linguagens.

Dependendo da rotina a ser executada, isso pode requerer várias consultas e atualizações na base, o que acarretará um maior consumo de recursos pela aplicação. No caso de aplicações web, isso se torna ainda mais visível, devido a maior quantidade de informações que precisam trafegar pela rede e de requisições ao servidor. Uma boa forma de contornar ou, ao menos atenuar,

esse consumo de recursos é transferir parte do processamento direto para o banco de dados, considerando que as máquinas servidoras geralmente têm configurações de *hardware* mais robustas. Entretanto, nada se pode garantir em relação às máquinas dos clientes.

A questão em que esbarramos é: como executar várias ações no banco de dados a partir de uma única instrução? A resposta para essa pergunta resume-se a *stored procedures*. Stored procedures são rotinas definidas no banco de dados, identificadas por um nome pelo qual podem ser invocadas. Este procedimento pode receber parâmetros, retornar valores e executar uma sequência de instruções, por exemplo: fazer update em uma tabela e, em sequência, inserir em outra e retornar um resultado de uma conta para sua aplicação.

7.1 Deixando o banco processar com stored procedures

Agora, em nosso sistema, temos a necessidade de criar um campo para armazenar o valor da comissão para cada venda. Esse valor será baseado na porcentagem de comissão que cada vendedor tem que ganhar, que estará cadastrada em um outro campo que também vamos criar na tabela de vendedores. Devemos criar esses dois campos utilizando o comando `alter table` que já aprendemos: o campo `n_porcvende` na tabela de vendedores e o `n_vcomvenda` na de vendas:

```
mysql> alter table comvende add n_porcvende float(10,2);
mysql> alter table comvenda add n_vcomvenda float(10,2);
```

Gerados os campos, vamos criar a nossa `storage procedure` que deverá buscar o valor da porcentagem de cada vendedor, realizar o processamento e, na sequência, fazer um *update* na coluna de valor da comissão na tabela de vendas.

Utilizaremos a `procedure` para fazer esse update, pois, em nosso cenário, já tínhamos criado o banco sem essas colunas e o nosso sistema já está em produção, isto é, estamos supondo que há pessoas utilizando-o. Isso ocorre a todo momento no desenvolvimento de sistemas. Há sempre a necessidade de novas colunas e novos processos. Por isso, devemos criar meios para adequar o nosso sistema. Veja como ficará:

```
mysql> delimiter $$
mysql>create procedure processa_comissionamento(
        in  data_inicial     date,
        in  data_final       date  ,
        out total_processado int )
        begin

            declare total_venda    float(10,2) default 0;
            declare venda          int         default 0;
            declare vendedor       int         default 0;
            declare comissao       float(10,2) default 0;
            declare valor_comissao float(10,2) default 0;
            declare aux            int         default 0;

            ## cursor para buscar os registros a serem
            ## processados entre a data inicial e data final
            ## e valor total de venda é maior que zero
            declare busca_pedido cursor for
                select n_numevenda,
                       n_totavenda,
                       n_numevende
                  from comvenda
                 where d_datavenda between data_inicial
                     and data_final
                      and n_totavenda > 0 ;

            ## abro o cursor
            open busca_pedido;

                ## inicio do loop
                vendas: LOOP

                ##recebo o resultado da consulta em cada variável
                fetch busca_pedido into venda, total_venda,
                vendedor;

                ## busco o valor do percentual de cada vendedor
                select n_porcvende
                  into comissao
```

```
  from comvende
 where n_numevende = vendedor;

## verifico se o percentual do vendedor é maior
## que zero logo após a condição deve ter o then
if (comissao > 0 ) then
    ## calculo o valor da comissao
    set valor_comissao  =
        ((total_venda * comissao)  / 100);

    ## faço o update na tabela comvenda com o
    ## valor da comissão
    update comvenda set
    n_vcomvenda = valor_comissao
    where n_numevenda = vendedor;
    commit;

## verifico se o percentual do vendedor é igual
## zero na regra do nosso sistema se o vendedor
## tem 0 ele ganha 0 porcento de comissão
elseif(comissao = 0) then

    update comvenda set n_vcomvenda = 0
    where n_numevenda = vendedor;
    commit;

## se ele não possuir registro no percentual de
## comissão ele irá ganhar 1 de comissão
## isso pela regra de negócio do nosso sistema
else
    set comissao = 1;
    set valor_comissao =
        ((total_venda * comissao)  / 100);

    update
        comvenda set n_vcomvenda = valor_comissao
    where n_numevenda = vendedor;
    commit;
## fecho o if
```

```
            end if;

            set comissao = 0;
            ##utilizo a variável aux para contar a quantidade
            set  aux      = aux +1 ;
        end loop vendas;
            ## atribuo o total de vendas para a variável de
            ## saída
        set total_processado = aux;
        ## fecho o cursor
        close busca_pedido;

        ##retorno o total de vendas processadas


        end$$

mysql>delimiter ;
```

Em nossa `procedure`, teremos parâmetros de entrada declarados com `in`, e de saída declarados com `out`, ambos na frente. Observe no código que, logo após o `begin`, eu faço a declaração das variáveis que utilizarei usando o `declare`. Já o `declare` que possui uma consulta embaixo é o que chamamos de `cursor`. Vamos utilizá-lo para recuperar registros por meio de consultas dentro de uma `procedure` ou `function`, como veremos mais à frente, e para fazer a busca das vendas que queremos processar.

Os passos para utilizar o `cursor` são:

- **open nome_cursor**: para abrir o `cursor`, que fará com que ele execute a consulta;
- **fetch nome_cursor into**: para atribuir o retorno do `cursor` a uma variável;
- **close nome_cursor**: quando terminarmos de utilizá-lo, devemos fechá-lo.

Observe na `procedure` que utilizo uma estrutura de controle: o `loop`. Ele permitirá, pela quantidade de registros, meu `cursor` abrir, fazer o cálculo

necessário para se obter o resultado da comissão e executar `update` na tabela. Vamos ver também o que ainda não tínhamos visto em SQL: a utilização de condições `if`, `elesif` e `else`.

Diferentemente de outras linguagens de programação que só possuem o `if` e `else`, no SQL também temos o `elesif`, que deve ser utilizado quando você precisa colocar mais uma condição antes do `else`. Além disso, não podemos esquecer que, ao encerrar a cláusula `if`, é preciso colocar `end if;`.

> **Observação**: como padrão, o `delimiter` do MySQL é o ponto e vírgula (`;`). Se você também estiver utilizando o prompt do MySQL para executar seus códigos, deve alterá-lo para outro caractere, uma vez que, no meio da `procedure`, temos vários ponto e vírgulas. Assim, precisamos dizer ao gerenciador onde é o final da nossa instrução. Em nosso exemplo, eu escolhi o `$$` para ser o delimitador temporário e, logo em seguida, voltei para o bom e velho ponto e vírgula.

Temos três parâmetros que devem ser passados ao executar a nossa `procedure`: dois de entrada e um de saída. O de saída poderá ser recuperado utilizando o `select`. Para executar este processo, utilizaremos o comando `call`, passaremos os parâmetros entre parênteses e, em seguida, recuperaremos o retorno, da seguinte maneira:

```
mysql> call
        processa_comissionamento('2015-01-01','2015-05-30' ,@a);
mysql> select @a;
```

Para você recriar essa `procedure` ou excluí-la, aplicando no SGBD novamente, você deve fazê-lo utilizando o mesmo comando para deletar. Em vez de `drop table`, será `drop procedure`. Assim, temos:

```
mysql> drop procedure processa_comissionamento;
```

Nossa primeira `procedure` ficou grande, pois queria colocar a maior quantidade de elementos possíveis: alguns novos e outros que aprendemos durante o projeto. Se você tiver a necessidade de fazer uma menor, apenas

deve ignorar os elementos de que você não necessita. A maior dificuldade quando estamos começando a programar em uma nova linguagem é juntar todos os elementos em um único bloco ou uma consulta, como estamos vendo no MySQL.

7.2 Processando e retornando com functions

Com as `procedures`, conseguimos realizar processamentos, e ainda, se quisermos, obter algum retorno. As funções são utilizadas especificamente para retornar algo. Podemos passar algum parâmetro com o mesmo tipo de declaração que fazemos na `procedure` e informar o tipo de retorno que teremos.

Se você quiser criar algo para ter algum retorno, aconselho a utilização de uma `function`, pois poderemos utilizá-las no meio de uma consulta, ao contrário da `procedure`, que temos que executar com um comando específico.

Vamos criar uma `function` para retornar o nome do cliente. Poderíamos fazer uma consulta por meio de um `join` para realizar esse retorno, o mesmo que fizemos no capítulo 5, quando queríamos retornar o nome do cliente da venda. Porém, imagine que você está fazendo uma consulta com muitas tabelas e `joins` em seu sistema. Ter uma função para ter o nome do cliente facilitaria. Então, mão na massa! Aliás, teclado!

```
mysql>delimiter $$
mysql> create function rt_nome_cliente(vn_numeclien int)
            returns varchar(50)

            begin

                declare nome varchar(50);

                select c_nomeclien  into nome
                  from comclien
                where n_numeclien = vn_numeclien;

                return nome;
```

```
            end $$
mysql> delimiter ;
```

Para fazer como uma consulta, só que passando um parâmetro entre parênteses:

```
mysql> ## estou passando como parâmetro o id do cliente igual a 1
mysql> select rt_nome_cliente(1);

+--------------------+
| rt_nome_cliente(1) |
+--------------------+
| AARONSON FURNITURE |
+--------------------+
1 row in set (0.02 sec)
```

E para utilizar a tabela `comvenda`, passamos o *id* do cliente de cada linha:

```
mysql> ##irei retornar o código da venda, nome do cliente e a
mysql> ##data da venda ordenando pelo nome e em seguida pela data
mysql> select c_codivenda,
              rt_nome_cliente(n_numeclien),
              d_datavenda
         from comvenda
        order by 2,3;

+-------------+-----------------------------+-------------+
| c_codivenda | rt_nome_cliente(n_numeclien) | d_datavenda |
+-------------+-----------------------------+-------------+
| 11          | AARONSON FURNITURE          | 2015-01-01  |
| 1           | AARONSON FURNITURE          | 2015-01-01  |
| 10          | AARONSON FURNITURE          | 2015-01-02  |
| 19          | AUTO WORKS                  | 2015-01-01  |
| 9           | AUTO WORKS                  | 2015-01-01  |
| 20          | AUTO WORKS                  | 2015-01-02  |
| 4           | GREAT AMERICAN MUSIC        | 2015-01-04  |
| 18          | HUGHES MARKETS              | 2015-01-04  |
| 8           | HUGHES MARKETS              | 2015-01-04  |
| 3           | KELSEY  NEIGHBOURHOOD       | 2015-01-03  |
| 13          | KELSEY  NEIGHBOURHOOD       | 2015-01-03  |
```

```
| 14          | KELSEY  NEIGHBOURHOOD         | 2015-01-04  |
| 15          | LIFE PLAN COUNSELLING         | 2015-01-01  |
| 5           | LIFE PLAN COUNSELLING         | 2015-01-01  |
| 2           | LITTLER                       | 2015-01-02  |
| 12          | LITTLER                       | 2015-01-02  |
| 6           | PRACTI-PLAN                   | 2015-01-02  |
| 16          | PRACTI-PLAN                   | 2015-01-02  |
| 7           | SPORTSWEST                    | 2015-01-03  |
| 17          | SPORTSWEST                    | 2015-01-03  |
+-------------+-------------------------------+-------------+
20 rows in set (0.00 sec)
```

7.3 Tabela dual

Em alguns bancos de dados, como Oracle, a cláusula `from` é obrigatória. Em consultas onde queremos retornar um único valor, devemos utilizar a tabela chamada `dual`. Por exemplo: o valor de uma constante, expressão, retornar o resultado de cálculos numéricos e de datas, ou seja, algo que não se origina de tabelas de dados comuns etc. Ela nada mais é que uma tabela, a qual possui uma coluna chamada `DUMMY` e um único registro com o valor X.

No MySQL, não temos essa obrigatoriedade, mas, se você quiser utilizá-la, não tem problema. O SGBD vai ignorá-la, já que o MySQL não tem a obrigatoriedade da cláusula `from`. Como exemplo, vamos utilizar a `function`: `rt_nome_cliente()`, mas só que agora utilizando a tabela `dual`.

```
mysql> select rt_nome_cliente(1) from dual;

+--------------------+
| rt_nome_cliente(1) |
+--------------------+
| AARONSON FURNITURE |
+--------------------+
1 row in set (0.02 sec)
```

É igual se tivéssemos feito:

```
mysql> select rt_nome_cliente(1);
```

```
+--------------------+
| rt_nome_cliente(1) |
+--------------------+
| AARONSON FURNITURE |
+--------------------+
1 row in set (0.02 sec)
```

Observe que o retorno é o mesmo. Você poderá utilizar como achar melhor, com ou sem o `dual`. Quando eu for utilizar consultas sem tabelas, farei das duas maneiras para você se acostumar.

Agora, você já pode criar uma `procedure` ou uma `function` para otimizar seu sistema. Use a criatividade, pois essas duas instruções podem ser muito úteis no dia a dia.

7.4 Automatizando o processo através de event scheduler

Criamos uma `procedure` para fazer o processamento das comissões. Porém, executar esse processamento pode se tonar uma atividade muito chata. Podemos agendar eventos para fazê-lo automática e periodicamente. Para isso, utilizamos o `event scheduler`.

Esses eventos agendados são bastante utilizados para fazer rotinas fora do horário de trabalho, principalmente de madrugada, ou se o sistema for utilizado 24 horas por dia. Escolha o horário de menor utilização, pois, geralmente, são processamentos grandes que requerem do servidor um amplo uso do processador de memória. Se realizado quando há muitas pessoas usando o sistema, pode atrapalhar a performance e até levar à queda do serviço de banco de dados. Mas nada impede de utilizá-lo para automatizar pequenas rotinas durante o dia.

Vamos programar a `procedure processa_comissionamento` para executar uma vez por semana. Por isso, utilizaremos `on schedule every 1 week`, que vai executar a primeira vez no dia '2015-03-01' ás 23:00 horas. Primeiro, devemos habilitar o `event_scheduler` em nosso SGBD, pois, por padrão, ele fica desabilitado. Abra o prompt e digite o comando:

```
mysql> set global event_scheduler = on;
```

```
Query OK, 0 rows affected (0.03 sec)
```

Nos parâmetros na chamada da `procedure`, utilizarei a função de data `current_date()` que retorna a data atual. Como o evento vai executar uma vez por semana, quero processar as vendas da semana. Logo, vou subtrair sete dias da data atual.

```
mysql> delimiter $$
mysql> create event processa_comissao
       on schedule every 1 week starts '2015-03-01 23:38:00'
       do
         begin
           call processa_comissionamento(
               current_date() - interval 7 day,
            current_date(), @a );
         end
mysql> $$
mysql> delimiter ;
```

Para vermos o resultado, vamos consultar as vendas desse período.

```
mysql> select c_codivenda Codigo,
              n_totavenda Total,
              n_vcomvenda Comissao
         from comvenda
        where
            d_datavenda between current_date() - interval 60 day
            and current_date();

+--------+----------+----------+
| Codigo | Total    | Comissao |
+--------+----------+----------+
| 2      | 12476.58 |  1743.99 |
| 3      | 16257.32 |     0.00 |
| 4      |  8704.55 |     0.00 |
| 6      |  6079.19 |     0.00 |
| 7      |  7451.26 |     0.00 |
| 8      | 15380.47 |     0.00 |
```

```
| 10       | 20315.07 |      0.00 |
| 12       | 11198.05 |      0.00 |
| 13       |  4967.84 |      0.00 |
| 14       |  7451.26 |      0.00 |
| 16       | 13502.34 |      0.00 |
| 17       | 22222.99 |      0.00 |
| 18       | 15465.69 |      0.00 |
| 20       |  6975.96 |      0.00 |
+--------+----------+----------+
14 rows in set (0.00 sec)
```

Podemos também executar os eventos com outras periodicidades, entre elas:

- **on schedule every 1 year**: uma vez por ano;
- **on schedule every 1 month**: uma vez por mês;
- **on schedule every 1 day**: uma vez ao dia;
- **on schedule every 1 hour**: uma vez por hora;
- **on schedule every 1 minute**: uma vez por minuto;
- **on schedule every 1 second**: uma vez por segundo.

Você pode utilizar o tempo que desejar. Eu exemplifiquei com o número `1`, mas você pode utilizar, por exemplo, `3 hour`.

Além de escolher quando ela começará, você também pode decidir quando parará de executar. Para exemplificar, vamos criar um evento para iniciar a nossa `procedure` a cada 10 minutos e parar depois de uma hora.

```
mysql> delimiter $$
mysql> create event processa_comissao_event
       on schedule every 10 minute
       starts current_timestamp
       ends current_timestamp + interval 30 minute
       do
        begin
```

```
            call processa_comissionamento(
                current_date() - interval 7 day,
             current_date(),
             @a);
        end
mysql> $$
Query OK, 0 rows affected (0.02 sec)

mysql> delimiter ;
```

Você pode utilizar os eventos de uma maneira versátil. Por exemplo: utilizá-los para realizar `insert`, `update`, `delete` e executar `procedures` e `functions`.

Ao criar um evento, ele fica habilitado automaticamente. Pode acontecer que, depois de um período, você não queira mais que o processo execute maquinalmente. Em vez de excluí-lo, você pode apenas desabilitá-lo com o seguinte comando:

```
mysql> alter event processa_comissao_event disable;

Query OK, 0 rows affected (0.00 sec)
```

E para habilitá-lo novamente:

```
mysql> alter event processa_comissao_event enable;

Query OK, 0 rows affected (0.00 sec)
```

Automatizar é preciso, pois desempenho é necessário. Essas duas coisas você consegue com `procedures`, `funtions` e `schedules`. Utilize-as com sabedoria e de uma forma que aumente a performance de seu sistema e sua produtividade, pois, assim, você ganhará no futuro em qualidade do sistema e em sua manutenção.

No repositório, você vai encontrar os arquivos `schedules.sql`, `procedures.sql` e `functions.sql`. Neles, há outros exemplos com outras variações. Agora, no próximo capítulo, vamos automatizá-lo ainda mais, através do SGBD.

Capítulo 8

Criando gatilhos

"Quando os fatos mudam, é preciso mudar as ideias."
– Tony Judt

8.1 Triggers nas rotinas

Criamos um processo para gerar a comissão, que pode ser executada manualmente ou com eventos programados. Porém, o nosso sistema não é muito grande e não tem uma numerosa quantidade de registros. Poderíamos, então, fazer esta operação em tempo real, no exato momento em que a venda é lançada. Conseguiremos isso através das *triggers*.

A *trigger* é um conjunto de operações que são executadas automaticamente quando uma alteração é feita em um registro que está relacionado a

uma tabela. Ela pode ser invocada antes ou depois de uma alteração em um `insert`, `update` ou `delete`, podendo haver até 6 triggers por tabela.

Alguns benefícios de sua utilização são:

- Verificar a integridade dos dados, pois é possível fazer uma verificação antes da inserção do registro;
- Contornar erros na regra de negócio do sistema no banco de dados;
- Utilizar como substituta para `event_scheduler`. Entretanto, ela não o substitui em processos que não são disparados a partir de uma tabela;
- Auditar as mudanças nas tabelas.

8.2 Triggers before insert e before update

Vamos colocar na prática. Como queremos realizar o cálculo da comissão automaticamente, devemos criar duas triggers: uma quando você insere uma nova venda e outra quando a atualizarmos. Utilizaremos as condições `before insert` (antes da inserção) e `before update` (antes da atualização). Além dessas duas, existem outras que mostrarei na sequência.

Para realizar a nossa operação, devemos consultar o percentual da comissão do cadastro de vendedores para gerar o cálculo. Colocando em prática o que já aprendemos, vamos criar uma `function` para ter esse percentual.

```
mysql> delimiter $$
mysql> create function rt_percentual_comissao(vn_n_numevende int)
       returns float
       deterministic
       begin
        declare percentual_comissao float(10,2);

        select n_porcvende
          into percentual_comissao
          from convende
         where n_numevende = vn_n_numevende;

        return percentual_comissao;
```

```
        end;
mysql> $$
mysql> delimiter ;
```

Vamos agora ao código para criar a trigger antes da inserção. Observe que vou utilizar o mesmo cálculo que usei na `procedure`.

```
mysql>  delimiter $$
mysql>  create trigger tri_vendas_bi
        before insert on comvenda
        for each row

        begin
          declare percentual_comissao float(10,2);

          ## busco o percentual de comissão que o vendedor deve
          ## receber
          select rt_percentual_comissao(new.n_numevende)
            into percentual_comissao;

          ## calculo a comissão
          set valor_comissao = ((total_venda * comissao)  / 100);

          ## recebo no novo valor de comissão
          set new.n_vcomvenda = valor_comissao;

        end
mysql> $$
mysql> delimiter ;
```

Agora, quando você inserir um novo registro na tabela `comvendas`, o cálculo do valor da comissão do vendedor vai ser realizado e o campo será preenchido.

Porém, o valor total da venda pode ser alterado e, caso ocorra a inserção ou retirada de um item dela, o valor da comissão a ser paga ao vendedor também mudará. Por isso, devemos criar mais uma `trigger` na tabela `comvendas` para fazer um `update` nesse valor para quando isso acontecer.

No lugar do `insert`, utilizaremos o `update`.

```
mysql>  delimiter $$
mysql>  create trigger tri_vendas_bu
        before update on comvenda
        for each row

        begin
        declare percentual_comissao float(10,2);
        declare total_venda         float(10,2);
        declare valor_comissao      float(10,2);

          ## No update, verifico se o valor total novo da venda
          ## é diferente do total anterior, pois se forem iguais,
          ## não há necessidade do cálculo
          if (old.n_totavenda <> new.n_totavenda) then
              select rt_percentual_comissao(new.n_numevende)
                into percentual_comissao;

              ## cálculo da comissão
              set
            valor_comissao = ((total_venda * comissao) / 100);

              ## recebo no novo valor de comissão
              set new.n_vcomvenda = valor_comissao;
          end if;
        end
mysql> $$
mysql> delimiter ;
```

Quando você alterar o valor total da venda, a comissão será gerada. Utilize os scripts que estão no arquivo `triggers.sql` no repositório para inserir e atualizar.

> **Dica**: observe que também padronizei os nomes das triggers, colocando no final do nome principal o seu tipo. Onde era `before update`, coloquei **bu** e para `before insert`, **bi**. Farei assim para os outros tipos também.

8.3 Triggers after insert e after update

Fizemos `triggers` para realizar o cálculo baseado no valor do total de vendas. Porém, estamos somando manualmente seus itens e inserindo no campo `n_totavenda`. Isso pode fazer com que algum erro ocorra, diferentemente de quando se insere a partir de uma aplicação; entretanto, podemos fazer com que esse cálculo seja realizado automaticamente, utilizando uma `trigger`.

Desta vez, vamos usar os tipos `after insert` (depois de inserir) e `after update` (depois de alterar) na tabela `comivenda` (itens da venda), para que, depois de inserir os produtos, o valor do seu total seja calculado e o campo `n_totavenda` seja atualizado.

```
mysql> delimiter $$
mysql> create trigger tri_vendas_ai
       after insert on comivenda
       for each row
       begin
         ## declaro as variáveis que utilizarei
         declare vtotal_itens float(10,2);
         declare vtotal_item  float(10,2);
         declare total_item   float(10,2);

         ## cursor para buscar os itens já registrados da venda
         declare busca_itens cursor for
           select n_totaivenda
             from comivenda
            where n_numevenda = new.n_numevenda;

          ## abro o cursor
          open busca_itens;
          ## declaro e inicio o loop
          itens : loop

              fetch busca_itens into total_item;
              ## somo o valor total dos itens(produtos)
              set vtotal_itens = vtotal_itens + total_item;

          end loop itens;
```

```
          close busca_itens;

          ## atualizo o total da venda
          update comvenda set n_totavenda = vtotal_itens
           where n_numevenda = new.n_numevenda;

         end
mysql> $$
mysql> delimiter ;
```

Com isso, ao inserir um registro na tabela de vendas, não precisamos preencher o campo `n_totavenda`, pois ele será preenchido automaticamente.

Agora temos a mesma situação que tínhamos anteriormente, pois a tabela de itens da venda pode ser atualizada e, se isso ocorrer, o valor de seu total ficará incorreto. Por isso, devemos criar uma trigger que o atualizará se o valor do item for alterado; mas somente na condição de o novo ser diferente do antigo. Esse não sendo o caso, não é necessário executar os cálculos.

```
mysql> delimiter $$
mysql> create trigger tri_ivendas_au
       after update on comivenda
       for each row
       begin
         ## declaro as variáveis que utilizarei
         declare vtotal_itens float(10,2);
         declare vtotal_item  float(10,2);
         declare total_item  float(10,2);

         ## cursor para buscar os itens já registrados da venda
           declare busca_itens cursor for
               select n_totaivenda
                 from comivenda
                where n_numevenda = new.n_numevenda;

         ## verifico se há necessidade de alteração
         ## faço somente se o novo valor for alterado
         if new.n_valoivenda <> old.n_valoivenda then
```

```
            ## abro o cursor
            open busca_itens;
              ## declaro e inicio o loop
              itens : loop

              fetch busca_itens into total_item;
              ## somo o valor total dos itens(produtos)
              set vtotal_itens = vtotal_itens + total_item;

            end loop itens;

            close busca_itens;

            ## atualizo o total da venda
            update comvenda set n_totavenda = vtotal_itens
             where n_numevenda = new.n_numevenda;

          end if;
        end
mysql> $$
mysql> delimiter ;
```

8.4 Triggers before delete e after delete

Agora, você pode estar com a seguinte dúvida: e se nós excluirmos um item de uma venda? Realmente, se isso ocorrer neste momento, o seu valor total estará incorreto, pois fizemos apenas as `triggers` para `insert` e `update`. Para corrigir este problema, vamos criar uma para o `delete` também.

```
mysql> delimiter $$
mysql> create trigger tri_ivendas_af
       after delete on comivenda
       for each row
       begin
         ## declaro as variáveis que utilizarei
         declare vtotal_itens float(10,2);
         declare vtotal_item  float(10,2);
         declare total_item  float(10,2);
```

```
        ## cursor para buscar os itens já registrados da venda
        declare busca_itens cursor for
          select n_totaivenda
            from comivenda
           where n_numevenda = old.n_numevenda;

        ## abro o cursor
        open busca_itens;

        ## declaro e inicio o loop
        itens : loop

          fetch busca_itens into total_item;
          ## somo o valor total dos itens(produtos)
          set vtotal_itens = vtotal_itens + total_item;

        end loop itens;

        close busca_itens;

        ## atualizo o total da venda
        update comvenda set n_totavenda = vtotal_itens
         where n_numevenda = old.n_numevenda;

       end
mysql> $$
mysql> delimiter ;
```

Agora, quando você excluir um item de uma venda, a trigger buscará os itens que ainda restam e fará o seu cálculo. No caso de vendas, eles serão todos automáticos.

Temos mais uma situação que podemos resolver utilizando a trigger: `on delete`. Lembra-se das questões de integridade de dados? Uma tabela que possui uma *foreign key* não pode deletar um registro sem antes fazer a exclusão do registro primário.

Portanto, utilizando uma trigger, ao fazer um `delete` na tabela vendas antes de excluí-la (`before delete`), vamos deletar os itens. Desta maneira,

respeitaremos a integridade, evitando erros no sistema e deletando os itens com um único comando. Você apenas precisará fazê-lo e a ela se encarregará do resto. Mãos ao teclado!

```
mysql> delimiter $$
mysql> create trigger tri_vendas_bf
       before delete on comvenda
       for each row
       begin
         ## declaro as variáveis que utilizarei
         declare vtotal_itens float(10,2);
         declare vtotal_item  float(10,2);
         declare total_item  float(10,2);

         ## verifico se há necessidade de alteração
         ## faço somente se o novo valor for alterado
         ## cursor para buscar os itens já registrados da venda
              declare busca_itens cursor for
             select n_totaivenda
                from comivenda
               where n_numevenda = old.n_numevenda;

         ## abro o cursor
         open busca_itens;
         ## declaro e inicio o loop
           itens : loop

             fetch busca_itens into total_item;
             ## somo o valor total dos itens(produtos)
             set vtotal_itens = vtotal_itens + total_item;

           end loop itens;

        close busca_itens;

        ## atualizo o total da venda
        delete from comivenda where n_numevenda = venda;
       end
mysql> $$
```

```
mysql> delimiter ;
```

Ao executarmos a seguinte instrução:

```
mysql> delete from comvenda where n_numevenda = 415;
```

Automaticamente, os itens serão deletados e a venda também!

8.5 Status das triggers

Por algumas razões, você pode querer não utilizar mais uma trigger. É possível escolher entre desabilitá-la ou excluí-la definitivamente. A vantagem da desabilitação é que você não precisará criá-la posteriormente, caso precise usá-la novamente. Por exemplo, se não quisermos mais que a comissão seja calculada automaticamente, podemos apenas invalidá-la.

Para desabilitar uma trigger, fazemos:

```
mysql> alter trigger tri_vendas_bi desable;
```

E se formos utilizar novamente o cálculo, habilitaremos novamente assim:

```
mysql> alter trigger tri_vendas_bi enable;
```

E para excluí-la:

```
mysql> drop trigger tri_vendas_bi;
```

Conhecemos mais uma ferramenta para nos auxiliar na automação de processos. Agora, conseguimos disparar processos automaticamente, aumentando a velocidade das operações do banco de dados, e utilizar mais um pouco da potência do nosso SGBD.

Capítulo 9

Obtendo performance e criando visões

"Sozinhos podemos ver pouco do futuro, porém o suficiente para darmos conta de que há muito que se fazer."
– Allan Turing

9.1 Ganhando performance com índices

Aprendemos a criar processos e a otimizá-los, utilizando a potência do SGBD. Fazemos isso pensando em sua performance e desempenho, pois, cada vez que a quantidade de dados do banco aumentar, você deverá pensar em métodos para fazer essa otimização.

Para nos auxiliar no aprimoramento das consultas, o MySQL nos fornece os chamados **índices**. Quando criamos as nossas tabelas, nós já criamos um índice de chave primária (*primary key*). Porém, ele não é utilizado para fazer essa otimização de performance, pois serve apenas para cuidar da integridade dos dados.

A medida inicial que devemos tomar para melhorar o tempo das consultas é a criação de índices para as tabelas. Se elas estão demorando para serem concluídas, as primeiras suspeitas são: ou eles não foram feitos ou estão mal criados. A adequação ou criação dos índices necessários pode resolver o problema na maioria das vezes; porém não sempre, pois seu banco de dados pode estar lento por outros motivos. Entretanto, como seu uso é o mais eficaz na questão de otimização, pode ser um desperdício de tempo tentar esse mesmo resultado por outros meios.

Mas o que acontece para ocorrer esse ganho de desempenho? Quando uma tabela não tem índices, os seus registros são desordenados e uma consulta terá que percorrer todos eles. Se adicionarmos um índice, uma nova tabela é gerada. A quantidade de registros desta nova é a mesma que a da original, a diferença é que eles são organizados. Isso implica que uma consulta percorrerá a tabela para encontrar os registros que casem com a sua condição e a busca é cessada quando um valor imediatamente maior é encontrado.

Se uma tabela possui 1000 registros, será, pelo menos, 100 vezes mais rápido do que ler todos eles sequencialmente. Porém, note que, se você precisar acessar quase todos eles, seria mais rápido acessá-los sequencialmente, porque evitaria acessos ao disco a cada verificação.

Criando index

Vamos demonstrar como se faz dos dois jeitos, começando pelo `create table`. Utilizarei como exemplo a nossa tabela de clientes, tanto na criação como depois dela. Criarei dois índices. Mais à frente, explicarei a escolha desses dois campos.

```
mysql> create table comclien(
                    n_numeclien int not null auto_increment,
                    c_codiclien varchar(10),
                    c_nomeclien varchar(200),
```

```
c_razaclien varchar(200),
d_dataclien date,
c_cnpjclien varchar(15),
c_foneclien varchar(15),
primary key (n_numeclien),
index idx_comclien_2(c_codiclien));
```

Agora, vamos ver como criar índices com `alter table`, uma vez que já criamos nossas tabelas no banco.

```
mysql> alter table comclien add
        index idx_comclien_3(c_razaclien);
mysql> alter table comclien add
        index idx_comclien_4(c_codiclien);
```

Quando utilizar

Você deve dar preferência para colunas que usamos para pesquisa, ordenação ou agrupamento, em cláusulas: `where`, `joins`, `order by` ou `group by`. Por isso, escolhi as colunas `n_numeclien`, `c_codiclien` e `c_razaclien`, já que são as que mais usamos durante o livro para fazer as consultas e buscas de registros. Porém, o fato de uma coluna aparecer na lista de colunas que serão exibidas em um `select` não a descarta de ser uma com `index`, pois ela pode estar na listagem e também estar na cláusula `where`, por exemplo.

Outro fator que você deve levar em consideração é a cardinalidade de uma coluna que é referenciada em outras tabelas, como *foreign key*, por ela conter uma grande quantidade de números distintos. Índices funcionam melhor em colunas com um alto número de cardinalidade relativa do que de registros da tabela; isto é, colunas que têm muitos valores únicos e poucos duplicados.

Quando não utilizar

Se uma coluna contém valores muito diferentes (por exemplo, a que guarda as idades), um índice vai diferenciar os registros rapidamente. Entretanto, ele não ajudará em uma que é usada para armazenar registros de gênero (sexo) e contém somente os valores `M` ou `F`. Se os registros têm, aproximadamente, o mesmo número de homens e mulheres, o índice percorrerá

quase metade dos registros, qualquer que seja o valor buscado. Com isso, podemos dizer que é melhor criar índices em colunas com grande variação de registros.

Até agora, citei as vantagens da utilização de `index`, pois não existem grandes desvantagens. Eu costumo dizer que os benefícios compensaram-nas. Com a criação dos índices nas tabelas, suas operações de `insert`, `update` e `delete` perderão velocidade. Isso ocorrerá, pois, ao realizar uma dessas alterações, a reordenação dos registros acontecerá. Não será nada perceptível ou que causará grande prejuízo de performance (por isso, digo que os benefícios são maiores). Em cada `insert` ou `update`, o SGBD ordenará os registros pelo `index` para suas consultas serem mais rápidas.

Nós podemos verificar os índices criados em uma tabela utilizando o mesmo `show` que usamos para ver o conteúdo de uma. Aqui, utilizaremos o `show indexes`, como no código na sequência.

```
mysql> show indexes from comclien;

+----------+------------+----------------+--------------
| Table    | Non_unique | Key_name       | Seq_in_index
+----------+------------+----------------+--------------
| comclien |          0 | PRIMARY        |            1
| comclien |          1 | idx_comclien_3 |            1
| comclien |          1 | idx_comclien_4 |            1
+----------+------------+----------------+--------------
+-------------+-----------+-------------+----------+--------
| Column_name | Collation | Cardinality | Sub_part | Packed
+-------------+-----------+-------------+----------+--------
| n_numeclien | A         |          10 |     NULL | NULL
| c_razaclien | A         |          10 |     NULL | NULL
| c_codiclien | A         |          10 |     NULL | NULL
+-------------+-----------+-------------+----------+--------
+------+------------+---------+---------------+
| Null | Index_type | Comment | Index_comment |
+------+------------+---------+---------------+
|      | BTREE      |         |               |
| YES  | BTREE      |         |               |
| YES  | BTREE      |         |               |
+------+------------+---------+---------------+
```

```
3 rows in set (0.02 sec)
```

O MySQL ainda possui outros tipos de índices. Outro mais comum é o `unique index`. Como o nome já diz, é um índice único que também serve para restringir a duplicação de dados. Ao criá-lo em uma coluna, você está dizendo ao SGBD que ele não pode aceitar registros duplicados lá. Portanto, tenha muito cuidado se for escolher trabalhar com esse tipo de índice.

Por exemplo, se criarmos um `unique index` na tabela de produtos no campo de `c_codiprodu`, não conseguiremos cadastrar dois deles com o mesmo código. Porém, é muito comum termos essa situação por conta de fornecedores diferentes. Analise bem sua regra de negócio antes de sair criando índices únicos.

Poderíamos criar um índice desse em nossa tabela de vendas no campo `c_codivenda`, para que não haja nenhum código duplicado. Desta maneira, fazemos:

```
mysql> alter table comvenda add unique
        index idx_comvenda_1(c_codivenda);

Query OK, 0 rows affected (0.92 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Vamos verificar se o índice foi criado.

```
mysql> show indexes from comvenda;

+----------+------------+----------------+--------------
| Table    | Non_unique | Key_name       | Seq_in_index
+----------+------------+----------------+--------------
| comvenda |          0 | PRIMARY        |            1
| comvenda |          0 | idx_comvenda_1 |            1
+----------+------------+----------------+--------------
+-------------+-----------+-------------+----------+--------
| Column_name | Collation | Cardinality | Sub_part | Packed
+-------------+-----------+-------------+----------+--------
| n_numevenda | A         |          20 |     NULL | NULL
| c_codivenda | A         |          20 |     NULL | NULL
```

```
+--------------+------------+--------------+-----------+--------
+------+------------+---------+---------------+
| Null | Index_type | Comment | Index_comment |
+------+------------+---------+---------------+
|      | BTREE      |         |               |
| YES  | BTREE      |         |               |
+------+------------+---------+---------------+


2 rows in set (0.00 sec)
```

Tente inserir uma venda com um código que já exista. Você verá que o banco retornará um erro, acusando sua existência e que esse campo possui um índice único.

Caso você tenha criado um `index` incorretamente, você poderá excluí-lo utilizando a seguinte instrução:

```
mysql> alter table comvenda drop index idx_comvenda_1;

Query OK, 0 rows affected (0.30 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Utilize índices em todas as tabelas e obtenha um pouco mais da potência do MySQL. Em nosso repositório, existe o arquivo `indices.sql`, no qual você pode ver a criação para as demais tabelas do projeto.

9.2 Views

Quando você trabalha com desenvolvimento de software e administração de dados em um SGBD, você escreve algumas determinadas consultas todos os dias e várias vezes. Muitas destas são derivadas de várias tabelas, o que nos dá um retrabalho ao montar todas aquelas `joins`.

Sabemos também que, toda vez que reescrevemos uma mesma consulta, conseguiremos os mesmos resultados de antes. Isso é uma tarefa séria, já que existem diversas formas de se escrever uma mesma consulta. Para amenizar essa situação e também pensando em performance e tempo economizado, podemos rapidamente transformar estas consultas em uma `view`. A partir

disso, ela permanecerá armazenada no servidor de bancos de dados em forma de tabela para que possamos consultá-la todas as vezes que precisarmos, sem ter que reescrevê-la.

Uma `view` é um objeto que pertence a um banco de dados, definida e baseada em declarações `selects`, retornando uma determinada visualização de dados de uma ou mais tabelas. Esses objetos são chamados, por vezes, de *virtual tables*, formados a partir de outras tabelas que, por sua vez, são denominadas de *based tables* ou ainda outras `Views`. Em alguns casos, estas são atualizáveis e podem ser alvos de declaração `insert`, `update` e `delete`, que, na verdade, modificam sua *based tables*.

Os benefícios de sua utilização, além dos já salientados, são:

- Uma `view` pode ser utilizada, por exemplo, para retornar um valor de apenas uma coluna na tabela;
- Também para promover restrições em dados para aumentar sua segurança e definir políticas de acesso em nível de tabela e coluna;
- Podem ser configuradas para mostrar colunas diferentes para diferentes usuários do banco de dados;
- Também podem ser usadas com um conjunto de tabelas unido a outros conjuntos de tabelas com a utilização de `joins`.

9.3 Criando Views

Uma consulta que já utilizamos algumas vezes durante o livro foi a da tabela de clientes junto com a de vendas. Sabendo disso, vamos criar uma `view` para essa consulta, para que possamos utilizar posteriormente, em vez de escrevê-la todas as vezes. Observe que sua criação será com `create or replace`, pois, se você quiser criá-la novamente, apenas precisa executar o código no SGBD e, assim, a nova `view` contendo o mesmo nome será reelaborada. Neste exemplo, eu a chamei de `clientes_vendas`.

```
mysql> create or replace view clientes_vendas as
           select c_razaclien,
```

```
                c_nomeclien,
                c_cnpjclien,
                c_codivenda,
                n_totavenda,
                d_datavenda
           from comclien,
                comvenda
           where comclien.n_numeclien = comvenda.n_numeclien
          order by 1;
```

Dessa maneira, podemos fazer uma consulta utilizando agora a nossa `view`, em vez da consulta convencional.

```
mysql> select * from clientes_vendas;

+-------------------------+----------------------
| c_razaclien             | c_nomeclien
+-------------------------+----------------------
| AARONSON FURNITURE LTD  | AARONSON FURNITURE
| LITTLER  LTDA           | LITTLER
| KELSEY  NEIGHBOURHOOD   | KELSEY  NEIGHBOURHOOD
| GREAT AMERICAN MUSIC    | GREAT AMERICAN MUSIC
| LIFE PLAN COUNSELLING   | LIFE PLAN COUNSELLING
| PRACTI-PLAN LTDA        | PRACTI-PLAN
| SPORTSWEST LTDA         | SPORTSWEST
| HUGHES MARKETS LTDA     | HUGHES MARKETS
| AUTO WORKS LTDA         | AUTO WORKS
| AARONSON FURNITURE LTD  | AARONSON FURNITURE
| AARONSON FURNITURE LTD  | AARONSON FURNITURE
| LITTLER  LTDA           | LITTLER
| KELSEY  NEIGHBOURHOOD   | KELSEY  NEIGHBOURHOOD
| KELSEY  NEIGHBOURHOOD   | KELSEY  NEIGHBOURHOOD
| LIFE PLAN COUNSELLING   | LIFE PLAN COUNSELLING
| PRACTI-PLAN LTDA        | PRACTI-PLAN
| SPORTSWEST LTDA         | SPORTSWEST
| HUGHES MARKETS LTDA     | HUGHES MARKETS
| AUTO WORKS LTDA         | AUTO WORKS
| AUTO WORKS LTDA         | AUTO WORKS
+-------------------------+----------------------
```

```
+--------------------+-------------+-------------+
| c_cnpjclien        | c_codivenda | n_totavenda |
+--------------------+-------------+-------------+
| 17.807.928/0001-85 | 1           |    25141.02 |
| 55.643.605/0001-92 | 2           |    12476.58 |
| 05.202.361/0001-34 | 3           |    16257.32 |
| 11.880.735/0001-73 | 4           |     8704.55 |
| 75.185.467/0001-52 | 5           |    13078.81 |
| 32.518.106/0001-78 | 6           |     6079.19 |
| 83.175.645/0001-92 | 7           |     7451.26 |
| 04.728.160/0001-02 | 8           |    15380.47 |
| 08.271.985/0001-00 | 9           |    13508.34 |
| 17.807.928/0001-85 | 10          |    20315.07 |
| 17.807.928/0001-85 | 11          |     8704.55 |
| 55.643.605/0001-92 | 12          |    11198.05 |
| 05.202.361/0001-34 | 13          |     4967.84 |
| 05.202.361/0001-34 | 14          |     7451.26 |
| 75.185.467/0001-52 | 15          |    10747.36 |
| 32.518.106/0001-78 | 16          |    13502.34 |
| 83.175.645/0001-92 | 17          |    22222.99 |
| 04.728.160/0001-02 | 18          |    15465.69 |
| 08.271.985/0001-00 | 19          |     4650.64 |
| 08.271.985/0001-00 | 20          |     6975.96 |
+--------------------+-------------+-------------+
20 rows in set (0.09 sec)
```

Atualizando views

Para você conseguir inserir, atualizar ou deletar um registro através de uma `view`, ela não pode possuir `joins` e funções agregadoras, como o `group by`. Vamos criar uma para a tabela de produtos.

```
mysql> create or replace view produtos as
             select n_numeprodu,
                    c_codiprodu,
                    c_descprodu,
                    n_valoprodu,
                    c_situprodu,
                    n_numeforne
```

```
                from comprodu;
```

Agora conseguiremos executar operações com ela. Da mesma maneira que aprendemos a fazer `insert` na tabela, faremos utilizando na `view`. Vamos exemplificar:

```
mysql> insert into produtos
          values (6,'0006','SMART WATCH','2412.98','A',1);

Query OK, 1 row affected (0.08 sec)
```

O mesmo vale para alteração e exclusão de registros.

```
mysql> update produtos set n_valoprodu = '1245.99'
        where n_numeprodu = 6;

Query OK, 1 row affected (0.13 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> commit;
Query OK, 0 rows affected (0.00 sec)

mysql> delete from produtos where n_numeprodu = 6;
Query OK, 1 row affected (0.09 sec)

mysql> commit;
Query OK, 0 rows affected (0.00 sec)a
```

Se você criou uma `view` que não utilizará novamente e precisa ser excluída, você deve fazer como se fosse deletar uma tabela. Utilizando o comando `drop`, da seguinte maneira:

```
mysql> drop view produtos;

Query OK, 0 rows affected (0.02 sec)
```

Conseguimos otimizar um pouco mais o nosso trabalho no banco de dados, melhorando a performance e criando `views` das nossas consultas mais comuns e de tabelas. Neste momento, você começa a perceber o quanto o

MySQL é potente em termos de ferramentas e flexibilidade. Você pode e deve extrair tudo o que ele tem a oferecer. Com o tempo, você vai otimizando seus próprios processos e, cada vez mais, extraindo, ao máximo, a potência do SGBD.

Capítulo 10

Criando, exportando e importando backups: ele poderá te salvar um dia

"Seja curioso. Leia de tudo. Tente coisas novas. O que as pessoas chamam de inteligência se resume a curiosidade."
– Aaron Swartz

10.1 Segurança dos seus dados

Imagine a quantidade de dados que o sistema sendo desenvolvido gera todos os dias. Agora, imagine que por consequência de uma falha de hardware, você possa perder um HD, no qual está seu banco de dados. O prejuízo seria muito

grande se você não possuísse uma cópia de segurança para substituir esse que estragou.

O *backup* de qualquer arquivo é importante, sendo o de bases de dados mais ainda. Não adianta você investir em segurança da informação para proteger sua rede e esquecer-se de proteger os dados com backups. Além de falhas de hardwares e de sistemas, também é bem comum usuários e desenvolvedores deletarem os dados por engano. Para atualizar a versão do MySQL ou trocar o banco, você também precisará de uma cópia da base. Por esses e outros motivos, devemos fazer backups sempre que possível: diariamente, semanalmente ou mensalmente, dependendo da sua preferência e necessidade.

No MySQl, você pode fazer um backup físico ou lógico. O primeiro consiste em fazer uma cópia das pastas de instalação do MySQL, que salvará toda a estrutura de arquivos de instalação, inclusive os dados. O segundo faz uma cópia da estrutura lógica, na qual estão inclusos os `scripts` de criação do banco de dados, das tabelas etc., além dos registros. Esta é a mais utilizada. Além dessas diferenças, o processo de backup pode ser realizado offline ou online.

Backup online

Os backups online ocorrem enquanto o servidor MySQL está em execução, para que as informações do banco de dados possam ser obtidas a partir do servidor.

Podemos destacar suas seguintes características:

- É menos intrusivo para outros clientes, que podem se conectar ao servidor MySQL durante o backup e pode ser capaz de acessar os dados, dependendo do que as operações necessitam para executar;
- Cuidados devem ser tomados para impor bloqueio apropriado para que as modificações de dados não ocorram de forma a comprometer a sua integridade.

Backup offline

Backups offline ocorrem enquanto o servidor está parado. Esta distinção entre eles também pode ser descrita como backups **quentes** *versus* backups

frios. Uma cópia de segurança **quente** é aquela em que o servidor continua funcionando, mas fica bloqueado contra modificação de dados enquanto você acessar arquivos de banco de dados externamente.

Também podemos destacar as seguintes características:

- Os clientes podem ser afetados negativamente, porque o servidor não está disponível durante o backup. Por essa razão, eles são, muitas vezes, tomados a partir de um servidor paralelo de replicação que pode ser tirado do ar, sem prejudicar a disponibilidade;
- O processo de cópia de segurança é simples, porque não existe qualquer possibilidade de interferência da atividade do cliente.

Essa questão de *on* ou *off* vai depender do cenário em que seu sistema estiver inserido, assim como várias que citei durante o livro. Você deve observar e resolver o que é melhor. Nós vamos aprender como fazer um backup lógico utilizando o console do seu sistema operacional. Como estamos em um ambiente de desenvolvimento, não importa se é offline ou online, pois não temos usuários acessando a nossa base de dados.

10.2 Criando backups

Diferentemente da instalação, que tem suas particularidades para cada sistema operacional, o comando da criação do backup será o mesmo. Utilizaremos variações do comando `mysqldump`. Vale lembrar que será apenas para exportar a base, pois, para importar, veremos mais à frente que é um comando diferente. Pode haver situações em que você apenas necessite exportar uma ou algumas tabelas do banco. Você tem a possibilidade de criar vários banco de dados e não fazer backup de todos. Por isso, também temos a possibilidade de exportar uma ou todas as bases de dados do banco.

Vamos criar um backup de todas as tabelas do banco do nosso projeto. Este criará um arquivo com a extensão `.sql`, que vai conter os scripts de criação e inserção de registros das tabelas exportadas. Vamos abrir o console e navegar até a pasta `bin` da instalação do MySQL.

```
mysql\bin> mysqldump -u root -p
             comercial > c:/bkp_tables_views.sql
```

Agora, abra o arquivo gerado em um editor de texto e você verá que o backup que foi gerado contém apenas as tabelas e os `inserts`. Mas e nossas `procedures`, `functions` e `triggers`? Para exportá-los juntos ao nosso arquivo, devemos especificar que também as queremos nessa exportação, com as instruções `--routines` para incluir a exportação das `procedures` e `functions`, e `--triggers` para incluir as `triggers`.

```
mysql\bin> mysqldump  -u root -p --routines --triggers
            comercial > c:/bkp_full.sql
```

Você também tem a possibilidade de exportar apenas uma ou várias tabelas. Você só precisa descrever quais delas você quer após o nome do banco de dados.

```
mysql\bin> mysqldump -u root -p  comercial
            comclien > c:/bkp_clien.sql
```

Até agora, nós criamos apenas um banco de dados. No entanto, há a possibilidade da criação de vários, como vimos durante a criação do nosso projeto. Aprendemos também como podemos exportar apenas um banco; mas também é possível criar um comando que exportará todos os bancos em apenas um arquivo. Faremos assim:

```
mysql\bin> mysqldump -u root -p --all-databases > c:/bkp_all.sql
```

Percebeu o quanto é fácil gerar um arquivo de backup? Você perderá apenas alguns minutos fazendo-o. Se ocorrer alguma falha e você não o tiver feito, vamos falar de dias, meses ou até anos perdido. Sem contar o dinheiro.

10.3 Importando backups

Para a importação, seguiremos passos parecidos com a exportação. Porém, antes, vamos criar um segundo banco de dados chamado `comercial2`. Nele, não nos preocuparemos com usuários, uma vez que utilizamos o usuário `root` para a exportação e também usaremos o arquivo `bkp_full.sql` que geramos na exportação.

```
mysql> create database comercial2;
```

Agora, vamos abrir o console e utilizar o comando:

```
mysql\bin> mysql -h localhost -u root -p -d
           comercial2 < c:/bkp_full.sql
```

Volte ao console do MySQL para testar se a importação foi feita com sucesso. Escolha o banco `comercial2` para utilizar e dê o comando para listar as tabelas. Se você seguiu todas as instruções, as listas delas deverão ser mostradas.

```
mysql> use comercial2;
mysql> show tables;
```

Agora você está apto para exportar e importar os seus dados em forma de arquivo de backup. Desta maneira, faça-o periodicamente pela segurança de seu sistema. O MySQL tem uma compactação muito boa para eles, pois não ficam muito grande, em comparação com o do Oracle, supondo o mesmo número de tabelas. Se pensarmos em espaço em disco para guardar todos os arquivos de backups, aconselho a fazê-lo, no mínimo, duas vezes ao dia.

Muitos desenvolvedores usam arquivos `.bat` juntamente com scripts específicos do sistema operacional que estão utilizando para fazer uma automatização desse processo no servidor de banco de dados. Você pode pesquisar sobre isso e adicioná-los aos nossos comandos de backup, como também pode usar ferramentas para fazê-lo. Em meu blog, fiz um post sobre esse processo no MySQL e sobre alguns instrumentos que você pode utilizar. O post pode ser acessado em: http://viniciuscdes.net/blog/2015/04/backup-mysql.

No próximo capítulo, você aprenderá a exportar e importar seus registros de uma maneira diferente: sem mexer na estrutura de seu banco, apenas neles.

Capítulo 11

MySQL avançado

"Se andarmos apenas por caminhos já traçados, chegaremos apenas aonde os outros chegaram."
– Alexander Graham Bell

Se você pesquisar sobre MySQL avançado em muitas bibliografias, poderá encontrar assuntos que englobam desde a criação de `procedures`, `event scheduler`, `view` e administração do SGBD. Essas partes que citei e que nós aprendemos durante o livro realmente fogem do básico. Entretanto, o que considero de fato avançado no MySQL é algo que não faz parte da rotina de desenvolvimento de software, esta que vai desde a criação de usuário ao monitoramento das atividades do SGBD.

Nós já vimos a criação de usuários lá no início do livro. Agora aprofundaremos em outros aspectos gerenciais, uma vez que estas funções fogem do dia

a dia do desenvolvedor e ficam a cargo do administrador do banco de dados. E, caso ele for fazê-las, será de forma esporádica.

11.1 Variáveis de sistema

Em todos os sistemas operacionais e em muitos softwares existem as variáveis de ambiente, que são configurações que podem ser alteradas. Aqui mostrarei como listá-las e alterá-las. Sua lista completa pode ser encontrada no site do manual oficial do MySQL: http://goo.gl/bGsSG3.

Primeiro, listarei todas as variáveis do MySQL para você saber como procurar uma e ver seu status. Vamos usar o comando `show`, o mesmo que utilizamos para elencar diversas coisas, como `tables`.

```
mysql> show variables;
```

Observe que a lista de variáveis fica um pouco bagunçada e, por isso, você pode acabar não encontrando aquela que deseja. O melhor é consultar como se fossem uma tabela, desde que você saiba o nome completo ou parcial. Vamos consultar as variáveis que possuem *table* em alguma parte do nome. Assim, ficaria:

```
mysql> show variables like '%table%';
```

Observe que ele listará as variáveis com os seus respectivos valores. Desta maneira, como disse anteriormente, podemos alterá-las.

No capítulo 2, falei brevemente sobre *case sensitive*, propriedade na qual faz o sistema operacional distinguir entre maiúsculas e minúsculas. Como você está mais familiarizado com as variáveis de ambiente, posso dizer que existe uma que executa esse controle, como um botão que liga e desliga essa propriedade do banco de dados.

A variável chama-se `lower_case_table_names`. Por padrão, no Linux possui valor 0, porque ele salva as tabelas como se elas estivessem escritas tanto em minúsculas ou maiúsculas, sendo *case sensitive*. Já no Windows, terá valor 1, pois as salvará em minúsculas e não será *case sensitive*; e no *MacOS* será 2, porque elas são salvas como estão escritas, não sendo também *case sensitive*. Logo, temos 3 status padrão para cada sistema operacional.

- **0**: tabelas serão criadas como estiverem escritas e o SGBD fará distinção entre maiúsculas e minúsculas;
- **1**: tabelas serão criadas em letras minúsculas e o SGBD não fará distinção entre maiúsculas e minúsculas;
- **2**: tabelas serão criadas como estiverem escritas e o SGBD não fará distinção entre maiúsculas e minúsculas.

> **Atenção**: se essa variável for alterada após você ter criado as tabelas, poderá ocasionar erros e falhas em seu sistema. As consultas já feitas em `procedures`, `triggers` etc. vão parar de funcionar. Então, cuidado! Aconselho utilizar o padrão, ou como fiz durante todo o nosso projeto: todas as criações em letra minúscula, pois, assim, independente do seu sistema operacional, o resultado será o mesmo.

Voltando para o que eu queria demonstrar: como mudar uma variável. Vamos alterar a `lower_case_table_names` para 2. Aconselho que, após mudar o seu valor, você altere novamente para o valor padrão do seu sistema operacional.

```
mysql> set lower_case_table_names = 2;
```

Não se preocupe em conhecer todas as variáveis ou configurá-las antes de utilizar o SGBD. Vá alterando-as conforme suas necessidades ou demandas dos projetos. A não ser que você venha a ser um administrador de banco de dados, pois então, seu papel será só preocupar-se com esses tipos de configurações.

11.2 Visualizando as conexões ativas

Quando criamos os agendamentos de eventos (`event scheduler`) e as `procedures`, descrevi que a melhor hora para executar um processo em uma grande quantidade de registros é quando o número de conexões for menor. Isso ocorre geralmente de madrugada. Entretanto, você pode listar todas as conexões ativas em seu SGBD a qualquer momento.

Para fazer esta consulta, também utilizaremos o comando `show`, que nos retornará valores, como: *PID* (número do processo da conexão), a quantidade de conexões do seu usuário à base, o processo que a conexão está executando no momento ou se ela está ociosa (*sleep*).

```
mysql> show processlist;
```

Quando estamos fazendo a administração de um banco de dados, preocupamo-nos em manter o seu desempenho e performance. Visualizando as conexões ativas, você pode se deparar com alguma que esteja travada ou até mesmo com algum usuário que esteja utilizando todas as disponíveis. Desse modo, você pode matar as conexões e processos que desejar. Primeiro, liste os processos e anote os números *ID*, pois utilizaremos o comando `kill` para matá-los, da seguinte maneira:

```
mysql> kill numero_id;
```

Muito cuidado ao utilizar a instrução `kill`, pois matar um processo incorreto pode gerar transtornos para seus usuários. Por exemplo, caso algum usuário do seu sistema esteja inserindo um novo registro no banco de dados ou emitindo uma venda, ao matar sua sessão, seu processo será interrompido antes que conclua a operação. Utilize-o com cautela.

11.3 Exportar e importar consultas para arquivos .csv e .txt

Para incluir registros no banco de dados, disponibilizei um arquivo que possuía uma série de scripts. No entanto, também poderíamos ter populado o banco de dados utilizando arquivos com as extensões `.csv` ou `.txt`, as mais utilizadas para se trabalhar com dados em exportações e importações.

Exportação

Como exemplo, vamos salvar todos os registros da tabela de clientes em um arquivo `.txt`. Na exportação, diremos para o SGBD que queremos salvar o arquivo na unidade `c:/` com o nome de `lista_clientes.txt`. Queremos separar cada registro por vírgula e limitar cada coluna com aspas simples.

```
mysql> select * from comclien
       into outfile 'c:/lista_clientes.txt'
       fields terminated by ','
       enclosed by '''';
```

Para você criar um arquivo `.csv`, apenas troque no código a extensão.

Observe que exportei todas as colunas da tabela. Você também pode escolher em sua consulta quais delas quer exportar. A exportação pode ser muito útil em seu dia a dia para extrair dados de uma forma rápida, sem precisar de uma aplicação. Com algumas linhas de código, você consegue extrair as informações de que tem necessidade. Além da consulta simples para realizar a exportação, você pode também criá-las com `joins`, funções de agrupamento (`group by`) etc.

Importação

A importação via arquivo pode ser muito útil em ações de popular dados em um banco de dados ou importá-los de outros sistemas ou bancos, como descrevi anteriormente. Da mesma forma que a exportação, ela não vai depender de uma aplicação ou de uma ferramenta.

Para realizar este processo, vamos criar uma tabela chamada `comuser`, que se referenciará aos usuários do nosso projeto.

```
mysql> create table comuser(
            n_numeuser int not null auto_increment,
            n_nomeuser varchar(100),
            n_nascuser date,
            primary key(n_numeuser));
```

Em nosso repositório, deixei um arquivo pronto para você utilizar nesta importação, o `import_user.txt`. Você mesmo pode também criar um arquivo com os registros, seguindo o padrão em que eles estão dispostos. Ele deve possuir as colunas que você quer popular.

Em nosso código, vamos descrever o nome do arquivo que queremos importar. Precisamos que ele esteja em alguma pasta em nosso computador. Coloque-o no diretório `c:/` ou onde preferir, contanto que não se esqueça de alterar o local no código.

Além de informarmos qual será o arquivo importado, devemos também dizer para qual tabela queremos fazer a importação. Sabendo disso, vamos indicar em qual delas queremos inserir os registros e com qual caractere eles estão separados e limitados, da mesma maneira como os exportamos.

```
mysql> load data infile 'c:/import_user.txt'
       into table comuser
       fields terminated by ','
       enclosed by '''';
```

11.4 Localizar uma coluna no seu banco

Algo muito útil é uma consulta para saber a quais tabelas um campo pertence. Particularmente, utilizo-a todos os dias, uma vez que trabalho em um cenário de mais de 1000 tabelas e fica difícil decorá-las. Por exemplo, em nosso projeto, criamos a coluna `n_numeclien` em duas tabelas. Desta maneira, é mais fácil saber em quais tabelas estão. Porém, como sempre friso, pense no longo prazo e tenha em mente que seu projeto crescerá.

Cada SGBD possui tabelas, nas quais são armazenados os objetos criados, tais como: outras tabelas, `views`, `procedures`, `triggers` etc. No MySQL, temos o `information_schema`, que é uma base de dados que possui as tabelas de metadados. São informações sobre o que temos criados no banco. A tabela que armazena as informações das outras criadas no SGBD é a `columns`. Para exemplificar, vamos utilizar o nosso banco de dados `comercial` e pesquisar quais delas possuem o campo `n_numeclien`.

```
mysql> select table_schema Banco_Dados,
              table_name tabela,
              column_name nome_coluna
         from information_schema.columns
        where table_schema = 'comercial'
          and column_name  = 'n_numeclien';
```

O resultado para a consulta que fizemos deve ser:

```
+---------------+------------+---------------+
| Banco_Dados   | tabela     | nome_coluna   |
+---------------+------------+---------------+
| comercial     | comclien   | n_numeclien   |
| comercial     | comvenda   | n_numeclien   |
+---------------+------------+---------------+
2 rows in set (0.02 sec)
```

Fig. 11.1: Busca de Colunas

Algo que faço para otimizar o mais meu tempo é salvar essa consulta em um arquivo com a extensão `.sql` e apenas chamá-la pelo prompt, em vez de digitá-la toda vez ou copiar e colá-la. Isso é algo que você pode fazer com todas as que são constantemente utilizadas.

A primeira coisa que devemos fazer é salvar a nossa consulta. Porém, antes vamos substituir o nome do banco e o da coluna que buscamos anteriormente para que essas variáveis sejam inseridas antes de executá-la. No lugar do nome do banco, vamos colocar `@banco` e no da coluna, `@coluna`. O símbolo de arroba diz para o SGBD que serão variáveis que serão recebidas em execução. Nossa consulta ficará assim:

```
mysql> select table_schema Banco_Dados,
              table_name tabela,
              column_name nome_coluna
         from information_schema.columns
        where table_schema = @banco
          and column_name  = @coluna;
```

Como possuo várias consultas que são utilizadas frequentemente, criei uma pasta chamada `scripts` no diretório `c:\` para armazenar os seus arquivos. Vamos nomeá-la como `busca_campo.sql`. Agora, abra o console do MySQL para atribuirmos os valores para as variáveis e, em seguida, fazermos a chamada.

```
mysql> set @banco = 'comercial';
mysql> set @coluna = 'n_numeclien';
mysql> source c:/scripts/campo_tabela.sql;
```

Você pode utilizar a criação de arquivos para qualquer tipo de consulta. Acho interessante você utilizar esse recurso para os comandos que aprendemos no início desse capítulo, pois seu uso ficará mais ágil. Em nosso repositório, há uma pasta `scripts`, na qual existem vários arquivos com as consultas que já utilizamos, inclusive uma chamada `info_banco.sql`. Ele, ao ser executado, trará várias informações sobre o seu banco, tais como: quantidade de *tables*, de *views* e outros objetos. Baixe e utilize-o em seu dia a dia!

11.5 Ferramentas para MySQL

Ferramentas com interface

Durante todo o livro, utilizei o próprio console do MySQL para fazer as operações. Ele é um pouco limitado, mas se você aprender a utilizá-lo, conseguirá usar qualquer outro tipo de ferramenta. As ferramentas com interface são como *IDEs*. Segue uma lista que pode deixar a utilização do MySQL mais eficiente.

- **Workbench**: é uma ferramenta tudo-em-um para tarefas, como: gerenciar seu servidor, escrever consultas, desenvolver procedimentos e trabalhar com diagramas. MySQL Workbench possui versão gratuita e comercial. Ela é mais do que suficiente para a maioria das necessidades. Você pode saber mais em http://www.mysql.com/products/workbench.

- **SQLyog**: SQLyog é uma das ferramentas visuais mais populares para MySQL com muitas características interessantes. É da mesma classe que o MySQL Workbench, mas algumas ferramentas tem funcionalidades que a outra não tem. Ela está disponível apenas para Windows e possui uma edição limitada gratuita, e uma cheia de recursos, que é paga. Mais informações em https://www.webyog.com.

- **phpMyAdmin**: é uma ferramenta de administração de servidores web mais utilizada. Oferece uma interface baseada em *browser* para seus servidores MySQL. Deve-se ter cuidado em utilizá-la, principalmente com a segurança de seu sistema, uma vez que, se o seu banco estiver

hospedado em um servidor web, o acesso ao `phpMyAdmin` possivelmente também dará acesso via web. Mais informações estão disponíveis em http://www.phpmyadmin.net.

Escolha aquela que mais lhe agrade e a que melhor se adapte, e tenha um ótimo desenvolvimento.

Ferramentas open source para monitoramento

Se você tornar-se responsável pelo desempenho do servidor de banco de dados e pelo monitoramento dos processos, você precisará de uma ferramenta para lhe auxiliar nessas tarefas. Existem várias para monitorar a infraestrutura de serviços, porém poucas ferramentas boas e gratuitas que sejam exclusivas para o MySQL. Por isso, aconselho apenas uma: o **Monyog**. Ele é excelente para *data base administrators* (DBAs) administrar seus banco de dados. Ele monitora o desempenho de consultas, espaço em disco, automatização de backups e muitas outras automatizações. Mais informações estão disponíveis em https://www.webyog.com/product.

Se você tem interesse por infraestrutura, pode pesquisar mais sobre estas excelentes ferramentas: *Nagios*, *Opsview* e *Icinga*. Além do MySQL, você também terá o monitoramento de todos os serviços do seu servidor.

Conforme aprendemos uma tecnologia, observamos que não existe uma separação muito bem definida sobre básico, intermediário e avançado. O que existe são categorias de determinados assuntos.

Durante o livro, vimos tópicos que são considerados avançados por alguns, mas fazem parte da rotina do desenvolvimento de software. Isso faz com que o assunto seja apenas algo a se praticar. Diferente de tópicos de administração do SGBD, que não farão parte do seu trabalho e que são uma categoria muito grande, nada impede que você se aprofunde no assunto de administração. Ou que também vá para a área de administração de banco de dados, se torne um DBA ou que continue no desenvolvimento e saiba bastante de administração. As possibilidades são infinitas.

Capítulo 12

Guia de consulta rápida

"A genialidade é 1% inspiração e 99% transpiração."
– Thomas Edson

12.1 O guia

Todos os guias rápidos que encontro são apenas palavras com significados, mas nunca um exemplo com a sintaxe completa. Por isso, resolvi colocar no final do livro algo que pudesse realmente lhe ajudar.

Aqui você encontrará a sintaxe de todos os comandos que aprendemos durante nosso projeto e também alguns novos. Utilize o guia depois da leitura do livro e de praticar todos os exemplos. Ele lhe ajudará a lembrar as sintaxe que você ainda não decorou.

12.2 Comandos ddl e dml

Comandos ddl

```
mysql> create table nome_tabela(
                    nome_coluna type,
                    primary key(coluna_primaria);

mysql> alter table nome_tabela add nome_coluna type;

mysql> alter table nome_tabela drop column nome_coluna;

mysql> alter table nome_tabela modify nome_coluna type;

mysql> drop table nome_tabela;

mysql> alter table nome_tabela add constraint primary key
        nome_constraint(nome_coluna);

mysql> alter table nome_tabela add constraint nome_constraint
            foreign key(nome_coluna)
                references nome_tabela_referenciada(
                nome_coluna_referenciada)
                    on delete no action
                    on update no action;

mysql> alter table nome_tabela drop constraint nome_constraint;
```

Comandos dml

```
mysql> insert into nome_tabela(nome_coluna)
            values (valores);

mysql> delete from nome_tabela
             where codicoes;

mysql> update nome_tabela set nome_coluna = valor
        where codicoes;
```

12.3 Tipos de dados

Escolha o tipo de texto que precisa para guardar cada informação.

Tipo texto

- **CHAR(tamanho)** : guarda um número fixo de caracteres. Pode conter letras, números e caracteres especiais. O tamanho deve ser declarado entre parênteses. Guarda até 255 caracteres.
- **VARCHAR(tamanho)**: possui as características do tipo `CHAR`, com a diferença de que, se você criar com mais de 255 caracteres, ele transforma-se no tipo `TEXT`. Ou seja, se for criar algum campo com mais de 255, já crie como `TEXT`.
- **TEXT**: guarda uma string com o tamanho máximo de 65.535 caracteres.
- **BLOB**: é o tipo de dado medido pela quantidade de bytes, em vez de pela quantidade de caracteres, conforme a maioria. Pode salvar por imagens, por exemplo, com o máximo de 65.535 bytes de arquivo.

Tipo numérico

- **TINYINT**: guarda números do tipo inteiro. Suporta de -128 até 127 caracteres.
- **SMALLINT**: guarda números do tipo inteiro. Suporta de -32768 até 32767 caracteres.
- **MEDIUMINT**: guarda números do tipo inteiro. Suporta de -8388608 até 8388607 caracteres.
- **INT(tamanho)**: guarda números inteiros. Suporta de -2147483648 até 2147483647 caracteres. O número máximo de caracteres pode ser especificado entre parênteses.
- **BIGINT**: guarda números do tipo inteiro. Suporta de -9223372036854775808 até 9223372036854775807 caracteres.

- **FLOAT(tamanho,decimal)**: guarda números `REAIS`. O número máximo de caracteres pode ser especificado entre parênteses. Deve-se especificar o tamanho inteiro e o tamanho numérico da coluna.
- **DOUBLE(tamanho,decimal)**: guarda números `REAIS`. O número máximo de caracteres pode ser especificado entre parênteses. Deve-se especificar o tamanho inteiro e o tamanho numérico da coluna. Esse tipo armazena uma quantidade maior de número do que os campos do tipo `FLOAT`.

Tipo data e tempo

- **DATE()**: tipo de campo que armazenará datas no formato YYYY-MM-DD, onde Y refere-se ao ano, M ao mês e D ao dia.
- **DATETIME()**: a combinação de data e tempo no formato YYYY-MM-DD HH:MI:SS.
- **TIME()**: armazena horas, minutos e segundos no formato HH:MI:SS.

12.4 Consultas

Consultas básicas

```
mysql> select * from nome_tabela;

mysql> select * from nome_tabela order by 1;

mysql> select * from nome_tabela order by 1;

mysql> select *
         from nome_tabela
        where codicoes
        order by 1;

mysql> select *
         from nome_tabela
```

```
        where nome_coluna = valor;

mysql> select *
         from nome_tabela
        where nome_coluna <> valor;

mysql> select *
         from nome_tabela
        where nome_coluna in (select nome_coluna
                                from nome_tabela2);

mysql> select *
         from nome_tabela
        where nome_coluna not in (select nome_coluna
                                    from nome_tabela2);

mysql> select campo1,
              campo2
         from nome_tabela
        group by campo1
        order by campo1;
```

Consultas com funções

Aprendemos algumas funções no capítulo 6 e outras durante o livro. Além das quais já aprendemos, incluí algumas novas.

Funções de agregação

```
## calcula o valor médio referente a uma coluna - avg()
mysql>select format(avg(campo_numerico),2) 'avarage price'
        from nome_tabela;

## para contar registros - count()
mysql> select count(*) from nome_tabela;

## verificar quantidade - having count()
mysql> select campo1, count(campo2)
         from nome_tabela
```

```
       having count(campo2) > 1;

## valor máximo e valor mínimo - max() / min()
mysql> select max(campo1), min(campo1)
         from nome_tabela;

## somar campos - sum()
mysql> select sum(campo1) from nome_tabela;


## selecionar registros distintos de uma coluna - distinct()
mysql> select distinct(campo1)
         from nome_tabela;
```

Funções numéricas

```
## retorna o arco co-seno de número acos(numero)
## ou null se x não estiver entre -1 e 1
mysql> select acos(numero) from nome_tabela;

## retorna o arco seno de número, asin(numero)
## ou null se número não estiver entre -1 e 1
mysql> select asin(numero) from nome_tabela;

## retorna o arco da tangente
mysql> select atan(numero) from nome_tabela;

## retorna o valor exponencial
mysql> select exp(numero) from nome_tabela;

## retorna o logaritmo natural base e
mysql> select log(3) from nome_tabela;

## retorna o logaritmo natural base 10
mysql> select log10(3) from nome_tabela;

## retorna a divisão de x por y.
mysql> select mod(x,y) from nome_tabela;
```

```
## retorna um valor aleatório
mysql> select rand(numero) from nome_tabela;

## arredondar números - round()
mysql> select round(campo_numerico)
         from nome_tabela;

## tirar a raiz quadrada de um número - sqrt()
mysql> select sqrt(campo_numerico)
         from nome_tabela;
```

Funções de string

```
## selecionar caracteres de uma string - substr
mysql> select substr(campo1,2)
         from nome_tabela;

## contar quantidade de caracteres em uma string - length(campo)
mysql>select length(campo,2)
         from nome_tabela;

## concatenar registros - concat() / concat_ws
mysql> select concat_ws(;,c_codiclien, c_razaclien, c_fantclien)
         from comclien;
        where c_razaclien like 'PEDR%';

## registros em mínusculo - lcase() / lower()
mysql> select lcase(c_razaclien)
         from comclien;

## registros em maiúsculo - ucase()
mysql> select ucase(c_razaclien)
         from comclien;

## completa uma string à direita com um caractere desejado
## na quantidade desejada
mysql> select rpad(string,10,' ')
         from nome_coluna;
```

Funcões de data

```
## retornar a diferença entre datas - datediff()
mysql>select ('2015-03-15','2015-03-17');

## converter de string para data - str_to_date()
mysql>select str_to_date('2013','%y');

## consulta a data e hora atual - now()
mysql>select now();

## retorna o dia do mês de uma data
mysql> select dayofmonth(data) from nome_tabela;

## retorna o valor numérico do dia da semana
mysql> select dayofweek(data) from nome_tabela;
```

12.5 Programando rotinas

Procedure

Para criar processos para o SGBD automatizar tarefas.

```
mysql> delimiter $$
mysql> create procedure
        processa_comissionamento(in|out|inout parametro tipo)
        begin

            instruções;

        end
mysql> $$
mysql> delimiter ;
```

Function

Crie para facilitar o seu dia a dia, retornando exatamente o que você precisa nas consultas.

```
mysql> create function rt_nome_cliente(vn_numeclien int)
            returns varchar(50)

        begin
            declare variavel_retorno datatype;

            instruções;

            return retorno;
        end
mysql> $$
mysql> delimiter ;
```

Event scheduler

Crie agendamentos para otimização de seus processos.

```
mysql>  create event processa_comissao
        on schedule every 1 [year|week|day|hour|minute|second]
        starts 'data_hora_qualquer'
        do
        begin

            instruções;
```

Trigger

Uma forma de disparar processos automaticamente através de alterações em registros de uma determinada tabela, como gatilhos.

```
mysql>  create or replace trigger nome_trigger
        [before|after insert|update|delete] on nome_tabela
        for each row

        begin
            intruções;
        end;

mysql> drop trigger nome_trigger;
```

12.6 Desempenho

Index

Dê mais desempenho às suas consultas.

```
mysql> alter table nome_tabela
         add index nome_index(nome_coluna);

mysql> alter table nome_tabela
         add unique index nome_index(nome_coluna);

mysql> show index from nome_tabela;

mysql> alter table nome_tabela drop index nome_index;
```

View

Não fique reescrevendo códigos repetitivos. Crie `views`.

```
mysql> create or replace view nome_view as
         select campos
           from tabelas
          where codições;

mysql> drop view nome_view;
```

12.7 Manutenção do banco

Backup

Crie backups com frequência. Ele poderá salvar seu sistema um dia!

```
## Exportando uma tabela
mysql\bin> mysqldump -u root -p
    comercial comclien > c:/bkp_clien.sql

## Exportando tudo
mysql\bin> mysqldump  -u root -p --routines --triggers
```

```
    comercial > c:/bkp_full.sql

## Importando tudo
mysql> mysql\bin> mysql -h localhost -u root -p -d
    comercial2 < c:/bkp_full.sql
```

Variáveis do ambiente

Para visualizarmos e alterarmos as variáveis do nosso ambiente. Use-as com cuidado.

```
## Listar as variáveis
mysql> show VARIABLES;

mysql> set nome_da_variavel = novo_valor;
```

Conexões

Visualizando e matando as conexões ativas em nosso banco de dados.

```
## Visualizando conexões ativas
mysql> show processlist;

## Matando conexões ativas
mysql> kill numero_PID;
```

Como esse guia, sua consulta ficará mais rápida. Também coloquei-o em meu blog para ajudá-lo quando estiver longe do livro. Para acessar, utilize o link http://www.viniciuscdes.net/blog/guiarapidomysql. Espero que o auxilie a escrever vários códigos legais.

Capítulo 13

Conclusão

"A jornada é a recompensa."
– Steve Jobs

13.1 O guia

Este livro o ajudou a escrever desde códigos simples até os mais avançados. Nos primeiros dois capítulos, tivemos uma introdução sobre o MySQL e iniciamos o planejamento do projeto que desenvolvemos ao seu decorrer. Nunca esqueça de planejar e documentar seus projetos. Já no capítulo 3, começamos a criar os códigos de nossas tabelas, conforme planejamos inicialmente.

Em seguida, quando já tínhamos nosso banco de dados e tabelas criados, faltavam-nos registros para serem manipulados. Portanto, no capítulo 4, os inserimos para popular nossas tabelas. Assim, começamos a brincar com eles por meio das consultas criadas no capítulo 5.

Aumentamos a dificuldade dos códigos e criamos consultas aprimoradas no capítulo 6, por meio do uso das `functions`. Aprendemos que o SGBD é uma poderosa ferramenta para processamento de cálculos e registros e que podemos usar esse poder para otimizar tarefas através das `procedures`, assunto tratado no capítulo 7. Além de aprender a programar automaticamente esses processos, no capítulo 8, vimos também como criar rotinas que eram disparadas por outras ações: as nossas `triggers`.

Quando há muitas informações e rotinas no banco de dados, devemos começar a pensar no aprimoramento da performance. No capítulo 9, mostrei como podemos otimizar as consultas através dos índices e também como não perder tempo criando a mesma diversas vezes por meio das `views`.

No capítulo 10, vimos o importante papel dos backups. Exportamos e importamos diversos deles do nosso banco de dados. Coloque-os em sua rotina de trabalho e nunca tenha problemas com perda de dados.

No capítulo 11, mostrei uma breve e importante introdução ao MySQL avançado e algumas instruções de seu gerenciamento. Para finalizar, no capítulo 12 criei um guia de consulta rápida com os principais comandos SQL, especialmente para auxiliá-lo no dia a dia!

Explorei ao máximo a aplicabilidade do MySQL no cotidiano do desenvolvedor com exemplos reais, para que você consiga desenvolver o que precisar. Contudo, como todas as linguagens de programação, ele também exige dedicação e treinamento para você se aperfeiçoar e otimizar seus códigos.

Não pare de estudar e programar. *Keep coding!*

Criei o fórum http://goo.gl/cdGLeY para que possamos manter contato e trocar experiências. Nele, poderemos discutir sobre banco de dados e, especialmente, sobre MySQL. Postem suas perguntas, dificuldades, sugestões e até soluções para os problemas do dia a dia. Estarei sempre à disposição para tirar dúvidas ou ajudar naqueles obstáculos que quase nos deixam loucos. Então, é isso. Até breve!